

Qwt User's Guide

5.2.2

Generated by Doxygen 1.7.3

Mon Aug 1 2011 11:28:26

Contents

1	Qwt - Qt Widgets for Technical Applications	1
1.1	License	1
1.2	Platforms	1
1.3	Screenshots	1
1.4	Downloads	1
1.5	Installation	2
1.6	Support	2
1.7	Related Projects	2
1.8	Language Bindings	2
1.9	Donations	2
1.10	Credits:	2
2	Qwt License, Version 1.0	3
3	INSTALL	11
4	Curve Plots	15
5	Scatter Plot	15
6	Spectrogram, Contour Plot	15
7	Histogram	15
8	Dials, Compasses, Knobs, Wheels, Sliders, Thermos	15
9	Deprecated List	15
10	Class Index	15
10.1	Class Hierarchy	15
11	Class Index	19
11.1	Class List	19
12	Class Documentation	22
12.1	QwtEventPattern::KeyPattern Class Reference	22
12.1.1	Detailed Description	23
12.2	QwtEventPattern::MousePattern Class Reference	23
12.2.1	Detailed Description	23
12.3	QwtAbstractScale Class Reference	23
12.3.1	Detailed Description	24
12.3.2	Constructor & Destructor Documentation	24
12.3.3	Member Function Documentation	25
12.4	QwtAbstractScaleDraw Class Reference	29
12.4.1	Detailed Description	31
12.4.2	Member Enumeration Documentation	31
12.4.3	Constructor & Destructor Documentation	31
12.4.4	Member Function Documentation	32
12.5	QwtAbstractSlider Class Reference	38

12.5.1 Detailed Description	40
12.5.2 Member Enumeration Documentation	40
12.5.3 Constructor & Destructor Documentation	40
12.5.4 Member Function Documentation	41
12.6 QwtAlphaColorMap Class Reference	52
12.6.1 Detailed Description	52
12.6.2 Member Enumeration Documentation	53
12.6.3 Constructor & Destructor Documentation	53
12.6.4 Member Function Documentation	53
12.7 QwtAnalogClock Class Reference	56
12.7.1 Detailed Description	60
12.7.2 Member Enumeration Documentation	60
12.7.3 Constructor & Destructor Documentation	61
12.7.4 Member Function Documentation	61
12.8 QwtArrayData Class Reference	85
12.8.1 Detailed Description	85
12.8.2 Constructor & Destructor Documentation	86
12.8.3 Member Function Documentation	86
12.9 QwtArrowButton Class Reference	88
12.9.1 Detailed Description	88
12.9.2 Constructor & Destructor Documentation	88
12.9.3 Member Function Documentation	89
12.10 QwtClipper Class Reference	91
12.10.1 Detailed Description	91
12.10.2 Member Function Documentation	91
12.11 QwtColorMap Class Reference	92
12.11.1 Detailed Description	93
12.11.2 Member Enumeration Documentation	94
12.11.3 Constructor & Destructor Documentation	94
12.11.4 Member Function Documentation	94
12.12 QwtCompass Class Reference	96
12.12.1 Detailed Description	100
12.12.2 Member Enumeration Documentation	100
12.12.3 Constructor & Destructor Documentation	101
12.12.4 Member Function Documentation	102
12.13 QwtCompassMagnetNeedle Class Reference	126
12.13.1 Detailed Description	127
12.13.2 Member Enumeration Documentation	127
12.13.3 Constructor & Destructor Documentation	127
12.13.4 Member Function Documentation	128
12.14 QwtCompassRose Class Reference	129
12.14.1 Detailed Description	130
12.14.2 Member Function Documentation	130
12.15 QwtCompassWindArrow Class Reference	131
12.15.1 Detailed Description	132
12.15.2 Member Enumeration Documentation	132
12.15.3 Constructor & Destructor Documentation	132
12.15.4 Member Function Documentation	133
12.16 QwtCounter Class Reference	134
12.16.1 Detailed Description	136

12.16.2 Member Enumeration Documentation	137
12.16.3 Constructor & Destructor Documentation	137
12.16.4 Member Function Documentation	138
12.17 QwtCPointerData Class Reference	147
12.17.1 Detailed Description	147
12.17.2 Constructor & Destructor Documentation	147
12.17.3 Member Function Documentation	148
12.18 QwtCurveFitter Class Reference	149
12.18.1 Detailed Description	150
12.18.2 Constructor & Destructor Documentation	150
12.18.3 Member Function Documentation	151
12.19 QwtData Class Reference	151
12.19.1 Detailed Description	152
12.19.2 Constructor & Destructor Documentation	152
12.19.3 Member Function Documentation	152
12.20 QwtDial Class Reference	154
12.20.1 Detailed Description	157
12.20.2 Member Enumeration Documentation	158
12.20.3 Constructor & Destructor Documentation	158
12.20.4 Member Function Documentation	159
12.21 QwtDialNeedle Class Reference	181
12.21.1 Detailed Description	181
12.21.2 Constructor & Destructor Documentation	182
12.21.3 Member Function Documentation	182
12.22 QwtDialScaleDraw Class Reference	183
12.22.1 Detailed Description	184
12.22.2 Member Enumeration Documentation	185
12.22.3 Constructor & Destructor Documentation	185
12.22.4 Member Function Documentation	185
12.23 QwtDialSimpleNeedle Class Reference	193
12.23.1 Detailed Description	194
12.23.2 Member Enumeration Documentation	194
12.23.3 Constructor & Destructor Documentation	195
12.23.4 Member Function Documentation	195
12.24 QwtDoubleInterval Class Reference	197
12.24.1 Detailed Description	198
12.24.2 Member Enumeration Documentation	198
12.24.3 Constructor & Destructor Documentation	199
12.24.4 Member Function Documentation	199
12.25 QwtDoubleRange Class Reference	205
12.25.1 Detailed Description	206
12.25.2 Constructor & Destructor Documentation	206
12.25.3 Member Function Documentation	206
12.26 QwtDynGridLayout Class Reference	212
12.26.1 Detailed Description	213
12.26.2 Constructor & Destructor Documentation	213
12.26.3 Member Function Documentation	213
12.27 QwtEventPattern Class Reference	219
12.27.1 Detailed Description	221
12.27.2 Member Enumeration Documentation	221

12.27.3 Constructor & Destructor Documentation	223
12.27.4 Member Function Documentation	223
12.28 QwtIntervalData Class Reference	227
12.28.1 Detailed Description	227
12.28.2 Constructor & Destructor Documentation	228
12.28.3 Member Function Documentation	228
12.29 QwtKnob Class Reference	229
12.29.1 Detailed Description	232
12.29.2 Member Enumeration Documentation	233
12.29.3 Constructor & Destructor Documentation	233
12.29.4 Member Function Documentation	233
12.30 QwtLegend Class Reference	251
12.30.1 Detailed Description	253
12.30.2 Member Enumeration Documentation	253
12.30.3 Constructor & Destructor Documentation	254
12.30.4 Member Function Documentation	254
12.31 QwtLegendItem Class Reference	259
12.31.1 Detailed Description	261
12.31.2 Member Enumeration Documentation	261
12.31.3 Constructor & Destructor Documentation	261
12.31.4 Member Function Documentation	262
12.32 QwtLegendItemManager Class Reference	269
12.32.1 Detailed Description	270
12.32.2 Constructor & Destructor Documentation	270
12.32.3 Member Function Documentation	271
12.33 QwtLinearColorMap Class Reference	271
12.33.1 Detailed Description	273
12.33.2 Member Enumeration Documentation	273
12.33.3 Constructor & Destructor Documentation	273
12.33.4 Member Function Documentation	274
12.34 QwtLinearScaleEngine Class Reference	278
12.34.1 Detailed Description	279
12.34.2 Member Enumeration Documentation	279
12.34.3 Member Function Documentation	279
12.35 QwtLog10ScaleEngine Class Reference	284
12.35.1 Detailed Description	286
12.35.2 Member Enumeration Documentation	286
12.35.3 Member Function Documentation	287
12.36 QwtMagnifier Class Reference	291
12.36.1 Detailed Description	293
12.36.2 Constructor & Destructor Documentation	293
12.36.3 Member Function Documentation	293
12.37 QwtMathMLTextEngine Class Reference	300
12.37.1 Detailed Description	301
12.37.2 Constructor & Destructor Documentation	301
12.37.3 Member Function Documentation	302
12.38 QwtMetricsMap Class Reference	303
12.38.1 Detailed Description	304
12.38.2 Member Function Documentation	304
12.39 QwtPainter Class Reference	305

12.39.1 Detailed Description	306
12.39.2 Member Function Documentation	306
12.40QwtPanner Class Reference	310
12.40.1 Detailed Description	311
12.40.2 Constructor & Destructor Documentation	311
12.40.3 Member Function Documentation	312
12.41QwtPicker Class Reference	316
12.41.1 Detailed Description	320
12.41.2 Member Enumeration Documentation	321
12.41.3 Constructor & Destructor Documentation	326
12.41.4 Member Function Documentation	327
12.42QwtPickerClickPointMachine Class Reference	344
12.42.1 Detailed Description	345
12.42.2 Member Enumeration Documentation	345
12.42.3 Member Function Documentation	345
12.43QwtPickerClickRectMachine Class Reference	346
12.43.1 Detailed Description	346
12.43.2 Member Enumeration Documentation	347
12.43.3 Member Function Documentation	347
12.44QwtPickerDragPointMachine Class Reference	347
12.44.1 Detailed Description	348
12.44.2 Member Enumeration Documentation	348
12.44.3 Member Function Documentation	349
12.45QwtPickerDragRectMachine Class Reference	349
12.45.1 Detailed Description	350
12.45.2 Member Enumeration Documentation	351
12.45.3 Member Function Documentation	351
12.46QwtPickerMachine Class Reference	351
12.46.1 Detailed Description	353
12.46.2 Member Enumeration Documentation	353
12.46.3 Constructor & Destructor Documentation	353
12.46.4 Member Function Documentation	353
12.47QwtPickerPolygonMachine Class Reference	354
12.47.1 Detailed Description	355
12.47.2 Member Enumeration Documentation	355
12.47.3 Member Function Documentation	355
12.48QwtPlainTextEngine Class Reference	356
12.48.1 Detailed Description	356
12.48.2 Constructor & Destructor Documentation	356
12.48.3 Member Function Documentation	357
12.49QwtPlot Class Reference	358
12.49.1 Detailed Description	362
12.49.2 Member Enumeration Documentation	362
12.49.3 Constructor & Destructor Documentation	363
12.49.4 Member Function Documentation	364
12.50QwtPlotCanvas Class Reference	385
12.50.1 Detailed Description	386
12.50.2 Member Enumeration Documentation	386
12.50.3 Constructor & Destructor Documentation	387
12.50.4 Member Function Documentation	387

12.51	QwtPlotCurve Class Reference	390
12.51.1	Detailed Description	394
12.51.2	Member Enumeration Documentation	395
12.51.3	Constructor & Destructor Documentation	397
12.51.4	Member Function Documentation	398
12.52	QwtPlotDict Class Reference	418
12.52.1	Detailed Description	419
12.52.2	Constructor & Destructor Documentation	419
12.52.3	Member Function Documentation	420
12.53	QwtPlotGrid Class Reference	421
12.53.1	Detailed Description	423
12.53.2	Member Enumeration Documentation	423
12.53.3	Constructor & Destructor Documentation	424
12.53.4	Member Function Documentation	424
12.54	QwtPlotItem Class Reference	437
12.54.1	Detailed Description	439
12.54.2	Member Enumeration Documentation	440
12.54.3	Constructor & Destructor Documentation	441
12.54.4	Member Function Documentation	441
12.55	QwtPlotLayout Class Reference	450
12.55.1	Detailed Description	451
12.55.2	Member Enumeration Documentation	451
12.55.3	Constructor & Destructor Documentation	452
12.55.4	Member Function Documentation	452
12.56	QwtPlotMagnifier Class Reference	459
12.56.1	Detailed Description	461
12.56.2	Constructor & Destructor Documentation	461
12.56.3	Member Function Documentation	461
12.57	QwtPlotMarker Class Reference	469
12.57.1	Detailed Description	472
12.57.2	Member Enumeration Documentation	472
12.57.3	Constructor & Destructor Documentation	473
12.57.4	Member Function Documentation	473
12.58	QwtPlotPanner Class Reference	487
12.58.1	Detailed Description	489
12.58.2	Constructor & Destructor Documentation	489
12.58.3	Member Function Documentation	490
12.59	QwtPlotPicker Class Reference	496
12.59.1	Detailed Description	500
12.59.2	Member Enumeration Documentation	500
12.59.3	Constructor & Destructor Documentation	505
12.59.4	Member Function Documentation	506
12.60	QwtPlotPrintFilter Class Reference	527
12.60.1	Detailed Description	529
12.60.2	Member Enumeration Documentation	529
12.60.3	Constructor & Destructor Documentation	529
12.60.4	Member Function Documentation	529
12.61	QwtPlotRasterItem Class Reference	531
12.61.1	Detailed Description	533
12.61.2	Member Enumeration Documentation	533

12.61.3 Constructor & Destructor Documentation	535
12.61.4 Member Function Documentation	535
12.62 QwtPlotRescaler Class Reference	546
12.62.1 Detailed Description	548
12.62.2 Member Enumeration Documentation	548
12.62.3 Constructor & Destructor Documentation	548
12.62.4 Member Function Documentation	549
12.63 QwtPlotScaleItem Class Reference	555
12.63.1 Detailed Description	557
12.63.2 Member Enumeration Documentation	557
12.63.3 Constructor & Destructor Documentation	558
12.63.4 Member Function Documentation	559
12.64 QwtPlotSpectrogram Class Reference	572
12.64.1 Detailed Description	574
12.64.2 Member Enumeration Documentation	575
12.64.3 Constructor & Destructor Documentation	576
12.64.4 Member Function Documentation	577
12.65 QwtPlotSvgItem Class Reference	593
12.65.1 Detailed Description	595
12.65.2 Member Enumeration Documentation	595
12.65.3 Constructor & Destructor Documentation	596
12.65.4 Member Function Documentation	597
12.66 QwtPlotZoomer Class Reference	606
12.66.1 Detailed Description	611
12.66.2 Member Enumeration Documentation	611
12.66.3 Constructor & Destructor Documentation	616
12.66.4 Member Function Documentation	618
12.67 QwtPolygonFData Class Reference	643
12.67.1 Detailed Description	644
12.67.2 Constructor & Destructor Documentation	644
12.67.3 Member Function Documentation	645
12.68 QwtRasterData Class Reference	646
12.68.1 Detailed Description	647
12.68.2 Member Enumeration Documentation	647
12.68.3 Constructor & Destructor Documentation	647
12.68.4 Member Function Documentation	648
12.69 QwtRichTextEngine Class Reference	651
12.69.1 Detailed Description	651
12.69.2 Constructor & Destructor Documentation	651
12.69.3 Member Function Documentation	652
12.70 QwtRoundScaleDraw Class Reference	653
12.70.1 Detailed Description	655
12.70.2 Member Enumeration Documentation	655
12.70.3 Constructor & Destructor Documentation	656
12.70.4 Member Function Documentation	656
12.71 QwtScaleArithmetic Class Reference	663
12.71.1 Detailed Description	664
12.71.2 Member Function Documentation	664
12.72 QwtScaleDiv Class Reference	666
12.72.1 Detailed Description	666

12.72.2 Member Enumeration Documentation	667
12.72.3 Constructor & Destructor Documentation	667
12.72.4 Member Function Documentation	667
12.73 QwtScaleDraw Class Reference	670
12.73.1 Detailed Description	672
12.73.2 Member Enumeration Documentation	672
12.73.3 Constructor & Destructor Documentation	673
12.73.4 Member Function Documentation	673
12.74 QwtScaleEngine Class Reference	685
12.74.1 Detailed Description	687
12.74.2 Member Enumeration Documentation	687
12.74.3 Constructor & Destructor Documentation	687
12.74.4 Member Function Documentation	688
12.75 QwtScaleMap Class Reference	692
12.75.1 Detailed Description	693
12.75.2 Constructor & Destructor Documentation	693
12.75.3 Member Function Documentation	693
12.76 QwtScaleTransformation Class Reference	696
12.76.1 Detailed Description	697
12.76.2 Constructor & Destructor Documentation	697
12.76.3 Member Function Documentation	697
12.77 QwtScaleWidget Class Reference	698
12.77.1 Detailed Description	700
12.77.2 Constructor & Destructor Documentation	700
12.77.3 Member Function Documentation	701
12.78 QwtSimpleCompassRose Class Reference	709
12.78.1 Detailed Description	710
12.78.2 Constructor & Destructor Documentation	710
12.78.3 Member Function Documentation	710
12.79 QwtSlider Class Reference	713
12.79.1 Detailed Description	716
12.79.2 Member Enumeration Documentation	716
12.79.3 Constructor & Destructor Documentation	717
12.79.4 Member Function Documentation	717
12.80 QwtSpline Class Reference	738
12.80.1 Detailed Description	739
12.80.2 Member Enumeration Documentation	739
12.80.3 Constructor & Destructor Documentation	739
12.80.4 Member Function Documentation	740
12.81 QwtSplineCurveFitter Class Reference	742
12.81.1 Detailed Description	743
12.81.2 Constructor & Destructor Documentation	744
12.81.3 Member Function Documentation	744
12.82 QwtSymbol Class Reference	745
12.82.1 Detailed Description	746
12.82.2 Member Enumeration Documentation	746
12.82.3 Constructor & Destructor Documentation	747
12.82.4 Member Function Documentation	747
12.83 QwtText Class Reference	751
12.83.1 Detailed Description	752

12.83.2 Member Enumeration Documentation	753
12.83.3 Constructor & Destructor Documentation	754
12.83.4 Member Function Documentation	755
12.84 QwtTextEngine Class Reference	762
12.84.1 Detailed Description	763
12.84.2 Constructor & Destructor Documentation	763
12.84.3 Member Function Documentation	764
12.85 QwtTextLabel Class Reference	765
12.85.1 Detailed Description	767
12.85.2 Constructor & Destructor Documentation	767
12.85.3 Member Function Documentation	767
12.86 QwtThermo Class Reference	770
12.86.1 Detailed Description	772
12.86.2 Constructor & Destructor Documentation	773
12.86.3 Member Function Documentation	773
12.87 QwtWheel Class Reference	786
12.87.1 Detailed Description	789
12.87.2 Member Enumeration Documentation	789
12.87.3 Constructor & Destructor Documentation	789
12.87.4 Member Function Documentation	790

1 Qwt - Qt Widgets for Technical Applications

The Qwt library contains GUI Components and utility classes which are primarily useful for programs with a technical background. Beside a 2D plot widget it provides scales, sliders, dials, compasses, thermometers, wheels and knobs to control or display values, arrays, or ranges of type double.

1.1 License

Qwt is distributed under the terms of the [Qwt License, Version 1.0](#).

1.2 Platforms

Qwt 5.x might be usable in all environments where you find [Qt](#). It is compatible with Qt 3.3.x and Qt 4.x, but the documentation is generated for Qt 4.x.

1.3 Screenshots

- [Curve Plots](#)
- [Scatter Plot](#)
- [Spectrogram, Contour Plot](#)
- [Histogram](#)

- [Dials, Compasses, Knobs, Wheels, Sliders, Thermos](#)

Screenshots are only available in the HTML docs.

1.4 Downloads

Stable releases, prereleases and snapshots are available at the [Qwt project page](#).

For getting a snapshot with all bugfixes for the latest 5.2 release:

```
svn co https://qwt.svn.sourceforge.net/svnroot/qwt/branches/qwt-5.2
```

For getting a development snapshot from the SVN repository:

```
svn co https://qwt.svn.sourceforge.net/svnroot/qwt/trunk/qwt
```

Qwt doesn't distribute binary packages, but today all major Linux distributors offer one. Note, that these packages often don't include the examples.

1.5 Installation

Have a look at the qwt.pro project file. It is prepared for building dynamic libraries in Win32 and Unix/X11 environments. If you don't know what to do with it, read the file [INSTALL](#) and/or Trolltechs [qmake](#) documentation. Once you have build the library you have to install all files from the lib, include and doc directories.

1.6 Support

- Mailing list

For all kind of Qwt related questions use 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 qwt-support@tigertal.de.

1.7 Related Projects

[QwtPolar](#), a polar plot widget.

[QwtPlot3D](#), an OpenGL 3D plot widget.

[QtiPlot](#), data analysis and scientific plotting tool, using [QwtPlot](#).

1.8 Language Bindings

[PyQwt](#), a set of Qwt Python bindings.

[Korundum/QtRuby](#), including a set of Qwt Ruby bindings.

1.9 Donations

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

1.10 Credits:

Authors:

Uwe Rathmann, Josef Wilgen (<= Qwt 0.2)

Project admin:

Uwe Rathmann <rathmann@users.sourceforge.net>

2 Qwt License, Version 1.0

Qwt License
Version 1.0, January 1, 2003

The Qwt library and included programs are provided under the terms of the GNU LESSER GENERAL PUBLIC LICENSE (LGPL) with the following exceptions:

1. Widgets that are subclassed from Qwt widgets do not constitute a derivative work.
2. Static linking of applications and widgets to the Qwt library does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared Qwt libraries, or link their applications or widgets against a user-supplied version of Qwt.

If you link the application or widget to a modified version of Qwt, then the changes to Qwt must be provided under the terms of the LGPL in sections 1, 2, and 4.

3. You do not have to provide a copy of the Qwt license with programs that are linked to the Qwt library, nor do you have to identify the Qwt license in your program or documentation as required by section 6 of the LGPL.

However, programs must still identify their use of Qwt. The following example statement can be included in user

documentation to satisfy this requirement:

[program/widget] is based in part on the work of
the Qwt project (<http://qwt.sf.net>).

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you. You must make sure that they, too, receive or can get the source
code. If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that
there is no warranty for the free library. Also, if the library is
modified by someone else and passed on, the recipients should know

that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

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.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the

Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free

programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.
```

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

```
You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

3 INSTALL

Introduction =====

Qwt uses qmake to build all its components and examples.
qmake is part of a Qt distribution.

qmake reads project files, that contain the options and rules how to build a certain project. A project file ends with the suffix "*.pro". Files that end with the suffix "*.pri" are included by the project files and contain definitions, that are common for several project files.

qwtconfig.pri is read by all project files of the Qwt package.
So the first step is to edit qwtconfig.pri to adjust it to your needs.

MathML Extension =====

Qwt/Qt4 supports the MathML render engine from the Qt solutions package, that is only available with a commercial Qt license.

You need a release of qtmmlwidget >= 2.1.
Copy the files qtmmlwidget.[cpp|h] to textengines/mathml.

Documentation =====

Qwt includes a class documentation, that is available in various formats:

- Html files
- PDF document
- Qt Compressed Help (*.qch) for the Qt assistant or creator.
You can load it "Edit Preferences" -> "Documentation" -> "Add..."
- Man pages (UNIX only)

A) Unix Qt3/Qt4 =====

```
qmake
make
make install
```

If you have installed a shared library it's path has to be known to the run-time linker of your operating system. On Linux systems read "man ldconfig" (or google for it). Another option is to use the LD_LIBRARY_PATH (on some systems LIBPATH is used instead, on MacOSX it is called DYLD_LIBRARY_PATH) environment variable.

If you only want to check the Qwt examples without installing something, you can set the LD_LIBRARY_PATH to the lib directory of your local build.

If you didn't enable autobuilding of the examples in qwtconfig.pri you have to build the examples this way:

```
cd examples
qmake
make
```

B) Win32/MSVC Qt3/Qt4
=====

Please read the qmake documentation how to convert your *.pro files into your development environment.

F.e MSVC with nmake:
qmake qwt.pro
nmake

If you didn't enable autobuilding of the examples in qwtconfig.pri you have to build the examples this way:

```
cd examples
qmake examples.pro
nmake
```

admin/msvc-qmake.bat helps users of Visual Studio users to generate makefiles or project files (.dsp for MSVC-6.0 or vcproj for MSVC.NET) for Qwt.

To generate makefiles, type: "admin\msvc-qmake"
To generate project files, type: "admin\msvc-qmake vc"

When you have built a Qwt DLL you need to add the following define to your compiler flags: QWT_DLL.

Windows doesn't like mixing of debug and release binaries. Most of the problems with using the Qwt designer plugin are because of trying to load a Qwt debug library into a designer release executable.

C) Win32/MinGW Qt4
=====

C1) Windows Shell

Start a Windows Shell, where Qt4 is initialized. (F.e. with "Programs->Qt by Trolltech ...->Qt 4.x.x Command Prompt").

```
qmake qwt.pro
make
```

If you didn't enable autobuilding of the examples in `qwtconfig.pri` you have to build the examples this way:

```
cd examples
qmake examples.pro
make
make install
```

C2) MSYS Shell Qt >= 4.3.0

Support for the MSYS Shell has been improved in Qt 4.3.0. Now building Qwt from the MSYS Shell works exactly like in UNIX or in the Windows Shell - or at least it should: because of a bug in Qt 4.3.0 you always have to do a `"qmake -r"`.

C3) MSYS Shell Qt < 4.3.0

For Qt < 4.3.0 you have to set the `MINGW_IN_SHELL` variable. `make` will run into errors with the `subdirs` target, that can be ignored (`make -i`).

```
export MINGW_IN_SHELL=1;

qmake
make -i
make -i install
```

If you didn't enable autobuilding of the examples in `qwtconfig.pri` you have to build the examples this way:

```
cd examples
qmake examples.pro
make -i
make -i install
```

C1-C3)

When you have built a Qwt DLL you need to add `QWT_DLL` to your compiler flags. If you are using `qmake` for your own builds this done by adding the following line to your profile: `"DEFINES += QWT_DLL"`.

Windows doesn't like mixing of debug and release binaries. Most of the problems with using the Qwt designer plugin are because of trying to load a Qwt debug library into a designer release executable.

D) MacOSX

Well, the Mac is only another Unix system. So read the instructions in A).

In the recent Qt4 releases the default target of `qmake` is to generate XCode project files instead of makefiles. So you might need to do the following:

```
qmake -spec macx-g++
...
```

D) Qtopia Core

I only tested Qwt with Qtopia Core in `qvfb` (Virtual Framebuffer Device) Emulator on my Linux box. To build Qwt for the emulator was as simple as

for a regular Unix build.

```
qmake
make
```

E) Qtopia (!= Qtopia Core)

I once compiled the Qwt library against Qtopia 4.2.0 successfully - but not more. It should be possible to build and install Qwt, but it's not done yet.

Good luck !

4 Curve Plots

5 Scatter Plot

6 Spectrogram, Contour Plot

/*!

7 Histogram

8 Dials, Compasses, Knobs, Wheels, Sliders, Thermos

9 Deprecated List

Member [QwtPlot::clear\(\)](#) Use [QwtPlotDeict::detachItems](#) instead

Class [QwtPlotPrintFilter](#) In Qwt 5.0 the design of [QwtPlot](#) allows/recommends writing individual [QwtPlotItems](#), that are not known to [QwtPlotPrintFilter](#). So this concept is outdated and [QwtPlotPrintFilter](#) will be removed/replaced in Qwt 6.x.

10 Class Index

10.1 Class Hierarchy

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

[QwtEventPattern::KeyPattern](#) [22](#)

[QwtEventPattern::MousePattern](#) [23](#)

QwtAbstractScale	23
QwtKnob	229
QwtSlider	713
QwtThermo	770
QwtAbstractScaleDraw	29
QwtRoundScaleDraw	653
QwtDialScaleDraw	183
QwtScaleDraw	670
QwtArrowButton	88
QwtClipper	91
QwtColorMap	92
QwtAlphaColorMap	52
QwtLinearColorMap	271
QwtCompassRose	129
QwtSimpleCompassRose	709
QwtCurveFitter	149
QwtSplineCurveFitter	742
QwtData	151
QwtArrayData	85
QwtCPointerData	147
QwtPolygonFData	643
QwtDialNeedle	181
QwtCompassMagnetNeedle	126
QwtCompassWindArrow	131
QwtDialSimpleNeedle	193
QwtDoubleInterval	197
QwtDoubleRange	205

QwtAbstractSlider	38
QwtDial	154
QwtAnalogClock	56
QwtCompass	96
QwtKnob	229
QwtSlider	713
QwtWheel	786
QwtCounter	134
QwtDynGridLayout	212
QwtEventPattern	219
QwtPicker	316
QwtPlotPicker	496
QwtPlotZoomer	606
QwtIntervalData	227
QwtLegend	251
QwtLegendItemManager	269
QwtPlotItem	437
QwtPlotCurve	390
QwtPlotGrid	421
QwtPlotMarker	469
QwtPlotRasterItem	531
QwtPlotSpectrogram	572
QwtPlotScaleItem	555
QwtPlotSvgItem	593
QwtMagnifier	291
QwtPlotMagnifier	459
QwtMetricsMap	303

QwtPainter	305
QwtPanner	310
QwtPlotPanner	487
QwtPickerMachine	351
QwtPickerClickPointMachine	344
QwtPickerClickRectMachine	346
QwtPickerDragPointMachine	347
QwtPickerDragRectMachine	349
QwtPickerPolygonMachine	354
QwtPlotCanvas	385
QwtPlotDict	418
QwtPlot	358
QwtPlotLayout	450
QwtPlotPrintFilter	527
QwtPlotRescaler	546
QwtRasterData	646
QwtScaleArithmetic	663
QwtScaleDiv	666
QwtScaleEngine	685
QwtLinearScaleEngine	278
QwtLog10ScaleEngine	284
QwtScaleMap	692
QwtScaleTransformation	696
QwtScaleWidget	698
QwtSpline	738
QwtSymbol	745
QwtText	751

QwtTextEngine	762
QwtMathMLTextEngine	300
QwtPlainTextEngine	356
QwtRichTextEngine	651
QwtTextLabel	765
QwtLegendItem	259

11 Class Index

11.1 Class List

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

QwtEventPattern::KeyPattern (A pattern for key events)	22
QwtEventPattern::MousePattern (A pattern for mouse events)	23
QwtAbstractScale (An abstract base class for classes containing a scale)	23
QwtAbstractScaleDraw (A abstract base class for drawing scales)	29
QwtAbstractSlider (An abstract base class for slider widgets)	38
QwtAlphaColorMap (QwtAlphaColorMap variies the alpha value of a color)	52
QwtAnalogClock (An analog clock)	56
QwtArrayData (Data class containing two QwtArray<double> objects)	85
QwtArrowButton (Arrow Button)	88
QwtClipper (Some clipping algos)	91
QwtColorMap (QwtColorMap is used to map values into colors)	92
QwtCompass (A Compass Widget)	96
QwtCompassMagnetNeedle (A magnet needle for compass widgets)	126
QwtCompassRose (Abstract base class for a compass rose)	129
QwtCompassWindArrow (An indicator for the wind direction)	131
QwtCounter (The Counter Widget)	134

QwtCPointerData (Data class containing two pointers to memory blocks of doubles)	147
QwtCurveFitter (Abstract base class for a curve fitter)	149
QwtData (QwtData defines an interface to any type of curve data)	151
QwtDial (QwtDial class provides a rounded range control)	154
QwtDialNeedle (Base class for needles that can be used in a QwtDial)	181
QwtDialScaleDraw (A special scale draw made for QwtDial)	183
QwtDialSimpleNeedle (A needle for dial widgets)	193
QwtDoubleInterval (A class representing an interval)	197
QwtDoubleRange (A class which controls a value within an interval)	205
QwtDynGridLayout (Lays out widgets in a grid, adjusting the number of columns and rows to the current size)	212
QwtEventPattern (A collection of event patterns)	219
QwtIntervalData (Series of samples of a value and an interval)	227
QwtKnob (The Knob Widget)	229
QwtLegend (The legend widget)	251
QwtLegendItem (A legend label)	259
QwtLegendItemManager (Abstract API to bind plot items to the legend)	269
QwtLinearColorMap (QwtLinearColorMap builds a color map from color stops)	271
QwtLinearScaleEngine (A scale engine for linear scales)	278
QwtLog10ScaleEngine (A scale engine for logarithmic (base 10) scales)	284
QwtMagnifier (QwtMagnifier provides zooming, by magnifying in steps)	291
QwtMathMLTextEngine (Text Engine for the MathML renderer of the Qt solutions package)	300
QwtMetricsMap (A Map to translate between layout, screen and paint device metrics)	303
QwtPainter (A collection of QPainter workarounds)	305
QwtPanner (QwtPanner provides panning of a widget)	310

QwtPicker (QwtPicker provides selections on a widget)	316
QwtPickerClickPointMachine (A state machine for point selections)	344
QwtPickerClickRectMachine (A state machine for rectangle selections)	346
QwtPickerDragPointMachine (A state machine for point selections)	347
QwtPickerDragRectMachine (A state machine for rectangle selections)	349
QwtPickerMachine (A state machine for QwtPicker selections)	351
QwtPickerPolygonMachine (A state machine for polygon selections)	354
QwtPlainTextEngine (A text engine for plain texts)	356
QwtPlot (A 2-D plotting widget)	358
QwtPlotCanvas (Canvas of a QwtPlot)	385
QwtPlotCurve (A plot item, that represents a series of points)	390
QwtPlotDict (A dictionary for plot items)	418
QwtPlotGrid (A class which draws a coordinate grid)	421
QwtPlotItem (Base class for items on the plot canvas)	437
QwtPlotLayout (Layout engine for QwtPlot)	450
QwtPlotMagnifier (QwtPlotMagnifier provides zooming, by magnifying in steps)	459
QwtPlotMarker (A class for drawing markers)	469
QwtPlotPanner (QwtPlotPanner provides panning of a plot canvas)	487
QwtPlotPicker (QwtPlotPicker provides selections on a plot canvas)	496
QwtPlotPrintFilter (A base class for plot print filters)	527
QwtPlotRasterItem (A class, which displays raster data)	531
QwtPlotRescaler (QwtPlotRescaler takes care of fixed aspect ratios for plot scales)	546
QwtPlotScaleItem (A class which draws a scale inside the plot canvas)	555
QwtPlotSpectrogram (A plot item, which displays a spectrogram)	572
QwtPlotSvgItem (A plot item, which displays data in Scalable Vector Graphics (SVG) format)	593

QwtPlotZoomer (QwtPlotZoomer provides stacked zooming for a plot widget)	606
QwtPolygonFData (Data class containing a single QwtArray<QwtDoublePoint> object)	643
QwtRasterData (QwtRasterData defines an interface to any type of raster data)	646
QwtRichTextEngine (A text engine for Qt rich texts)	651
QwtRoundScaleDraw (A class for drawing round scales)	653
QwtScaleArithmetic (Arithmetic including a tolerance)	663
QwtScaleDiv (A class representing a scale division)	666
QwtScaleDraw (A class for drawing scales)	670
QwtScaleEngine (Base class for scale engines)	685
QwtScaleMap (A scale map)	692
QwtScaleTransformation (Operations for linear or logarithmic (base 10) transformations)	696
QwtScaleWidget (A Widget which contains a scale)	698
QwtSimpleCompassRose (A simple rose for QwtCompass)	709
QwtSlider (The Slider Widget)	713
QwtSpline (A class for spline interpolation)	738
QwtSplineCurveFitter (A curve fitter using cubic splines)	742
QwtSymbol (A class for drawing symbols)	745
QwtText (A class representing a text)	751
QwtTextEngine (Abstract base class for rendering text strings)	762
QwtTextLabel (A Widget which displays a QwtText)	765
QwtThermo (The Thermometer Widget)	770
QwtWheel (The Wheel Widget)	786

12 Class Documentation

12.1 QwtEventPattern::KeyPattern Class Reference

A pattern for key events.

```
#include <qwt_event_pattern.h>
```

Public Member Functions

- **KeyPattern** (int k=0, int st=Qt::NoButton)

Public Attributes

- int **key**
- int **state**

12.1.1 Detailed Description

A pattern for key events.

12.2 QwtEventPattern::MousePattern Class Reference

A pattern for mouse events.

```
#include <qwt_event_pattern.h>
```

Public Member Functions

- **MousePattern** (int btn=Qt::NoButton, int st=Qt::NoButton)

Public Attributes

- int **button**
- int **state**

12.2.1 Detailed Description

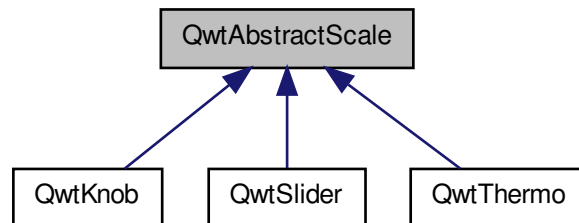
A pattern for mouse events.

12.3 QwtAbstractScale Class Reference

An abstract base class for classes containing a scale.

```
#include <qwt_abstract_scale.h>
```


Inheritance diagram for QwtAbstractScale:



Public Member Functions

- bool [autoScale](#) () const
- [QwtAbstractScale](#) ()
- const [QwtScaleEngine](#) * [scaleEngine](#) () const
- [QwtScaleEngine](#) * [scaleEngine](#) ()
- const [QwtScaleMap](#) & [scaleMap](#) () const
- int [scaleMaxMajor](#) () const
- int [scaleMaxMinor](#) () const
- void [setAutoScale](#) ()
- void [setScale](#) (const [QwtDoubleInterval](#) &, double step=0.0)
- void [setScale](#) (const [QwtScaleDiv](#) &s)
- void [setScale](#) (double vmin, double vmax, double step=0.0)
- void [setScaleEngine](#) ([QwtScaleEngine](#) *)
- void [setScaleMaxMajor](#) (int ticks)
- void [setScaleMaxMinor](#) (int ticks)
- virtual ~[QwtAbstractScale](#) ()

Protected Member Functions

- const [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) () const
- [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) ()
- void [rescale](#) (double vmin, double vmax, double step=0.0)
- virtual void [scaleChange](#) ()
- void [setAbstractScaleDraw](#) ([QwtAbstractScaleDraw](#) *)

12.3.1 Detailed Description

An abstract base class for classes containing a scale. [QwtAbstractScale](#) is used to provide classes with a [QwtScaleDraw](#), and a [QwtScaleDiv](#). The [QwtScaleDiv](#) might be set explicitly or calculated by a [QwtScaleEngine](#).

12.3.2 Constructor & Destructor Documentation

12.3.2.1 QwtAbstractScale::QwtAbstractScale ()

Constructor

Creates a default [QwtScaleDraw](#) and a [QwtLinearScaleEngine](#). Autoscaling is enabled, and the stepSize is initialized by 0.0.

12.3.2.2 QwtAbstractScale::~~QwtAbstractScale () [virtual]

Destructor.

12.3.3 Member Function Documentation

12.3.3.1 const QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () const [protected]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.3.3.2 QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () [protected]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.3.3.3 bool QwtAbstractScale::autoScale () const**Returns**

true if autoscaling is enabled

12.3.3.4 void QwtAbstractScale::rescale (double *vmin*, double *vmax*, double *stepSize* = 0.0) [protected]

Recalculate the scale division and update the scale draw.

Parameters

<i>vmin</i>	Lower limit of the scale interval
<i>vmax</i>	Upper limit of the scale interval
<i>stepSize</i>	Major step size

See also

[scaleChange\(\)](#)

12.3.3.5 void QwtAbstractScale::scaleChange () [protected, virtual]

Notify changed scale.

Dummy empty implementation, intended to be overloaded by derived classes

Reimplemented in [QwtSlider](#), and [QwtThermo](#).

12.3.3.6 const QwtScaleEngine * QwtAbstractScale::scaleEngine () const**Returns**

Scale engine

See also

[setScaleEngine\(\)](#)

12.3.3.7 QwtScaleEngine * QwtAbstractScale::scaleEngine ()

Returns

Scale engine

See also

[setScaleEngine\(\)](#)

12.3.3.8 const QwtScaleMap & QwtAbstractScale::scaleMap () const**Returns**

[abstractScaleDraw\(\)](#)->[scaleMap\(\)](#)

12.3.3.9 int QwtAbstractScale::scaleMaxMajor () const**Returns**

Max. number of major tick intervals The default value is 5.

12.3.3.10 int QwtAbstractScale::scaleMaxMinor () const**Returns**

Max. number of minor tick intervals The default value is 3.

12.3.3.11 void QwtAbstractScale::setAbstractScaleDraw (QwtAbstractScaleDraw * *scaleDraw*) [protected]

Set a scale draw.

scaleDraw has to be created with new and will be deleted in ~QwtAbstractScale or the next call of setAbstractScaleDraw.

12.3.3.12 void QwtAbstractScale::setAutoScale ()

Advise the widget to control the scale range internally.

Autoscaling is on by default.

See also

[setScale\(\)](#), [autoScale\(\)](#)

12.3.3.13 void QwtAbstractScale::setScale (double *vmin*, double *vmax*,
double *stepSize* = 0.0)

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>vmin</i>	lower limit of the scale interval
<i>vmax</i>	upper limit of the scale interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.3.3.14 void QwtAbstractScale::setScale (const QwtScaleDiv & *scaleDiv*)

Specify a scale.

Disable autoscaling and define a scale by a scale division

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

See also

[setAutoScale\(\)](#)

12.3.3.15 void QwtAbstractScale::setScale (const QwtDoubleInterval &
interval, double *stepSize* = 0.0)

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>interval</i>	Interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.3.3.16 void QwtAbstractScale::setScaleEngine (QwtScaleEngine * *scaleEngine*)

Set a scale engine.

The scale engine is responsible for calculating the scale division, and in case of auto scaling how to align the scale.

scaleEngine has to be created with `new` and will be deleted in `~QwtAbstractScale` or the next call of `setScaleEngine`.

12.3.3.17 void QwtAbstractScale::setScaleMaxMajor (int *ticks*)

Set the maximum number of major tick intervals.

The scale's major ticks are calculated automatically such that the number of major intervals does not exceed *ticks*. The default value is 5.

Parameters

<i>ticks</i>	maximal number of major ticks.
--------------	--------------------------------

See also

[QwtAbstractScaleDraw](#)

12.3.3.18 void QwtAbstractScale::setScaleMaxMinor (int *ticks*)

Set the maximum number of minor tick intervals.

The scale's minor ticks are calculated automatically such that the number of minor intervals does not exceed *ticks*. The default value is 3.

Parameters

<i>ticks</i>

See also

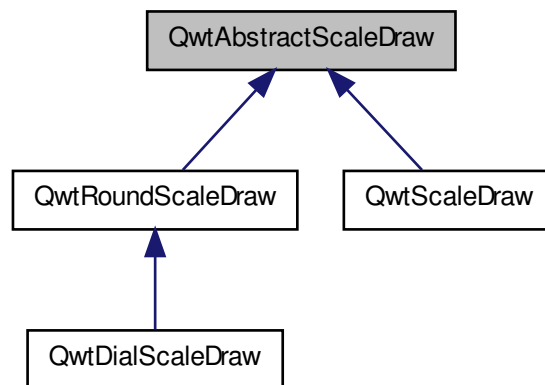
[QwtAbstractScaleDraw](#)

12.4 QwtAbstractScaleDraw Class Reference

A abstract base class for drawing scales.

```
#include <qwt_abstract_scale_draw.h>
```

Inheritance diagram for QwtAbstractScaleDraw:



Public Types

- enum [ScaleComponent](#) {
 Backbone = 1,
 Ticks = 2,
 Labels = 4 }

Public Member Functions

- virtual void [draw](#) (QPainter *, const QPalette &) const
- void [enableComponent](#) ([ScaleComponent](#), bool enable=true)
- virtual int [extent](#) (const QPen &, const QFont &) const =0
- bool [hasComponent](#) ([ScaleComponent](#)) const
- virtual [QwtText](#) [label](#) (double) const
- int [majTickLength](#) () const
- const [QwtScaleMap](#) & [map](#) () const
- int [minimumExtent](#) () const
- [QwtAbstractScaleDraw](#) & [operator=](#) (const [QwtAbstractScaleDraw](#) &)
- [QwtAbstractScaleDraw](#) (const [QwtAbstractScaleDraw](#) &)

- [QwtAbstractScaleDraw](#) ()
- const [QwtScaleDiv](#) & [scaleDiv](#) () const
- [QwtScaleMap](#) & [scaleMap](#) ()
- void [setMinimumExtent](#) (int)
- void [setScaleDiv](#) (const [QwtScaleDiv](#) &s)
- void [setSpacing](#) (int margin)
- void [setTickLength](#) ([QwtScaleDiv::TickType](#), int length)
- void [setTransformation](#) ([QwtScaleTransformation](#) *)
- int [spacing](#) () const
- int [tickLength](#) ([QwtScaleDiv::TickType](#)) const
- virtual [~QwtAbstractScaleDraw](#) ()

Protected Member Functions

- virtual void [drawBackbone](#) (QPainter *painter) const =0
- virtual void [drawLabel](#) (QPainter *painter, double value) const =0
- virtual void [drawTick](#) (QPainter *painter, double value, int len) const =0
- void [invalidateCache](#) ()
- const [QwtText](#) & [tickLabel](#) (const QFont &, double value) const

12.4.1 Detailed Description

A abstract base class for drawing scales. [QwtAbstractScaleDraw](#) can be used to draw linear or logarithmic scales.

After a scale division has been specified as a [QwtScaleDiv](#) object using [QwtAbstractScaleDraw::setScaleDiv\(const QwtScaleDiv &s\)](#), the scale can be drawn with the [QwtAbstractScaleDraw::draw\(\)](#) member.

12.4.2 Member Enumeration Documentation

12.4.2.1 enum QwtAbstractScaleDraw::ScaleComponent

Components of a scale

- Backbone
- Ticks
- Labels

See also

[enableComponent\(\)](#), [hasComponent](#)

12.4.3 Constructor & Destructor Documentation

12.4.3.1 QwtAbstractScaleDraw::QwtAbstractScaleDraw ()

Constructor.

The range of the scale is initialized to [0, 100], The spacing (distance between ticks and labels) is set to 4, the tick lengths are set to 4, 6 and 8 pixels

12.4.3.2 QwtAbstractScaleDraw::QwtAbstractScaleDraw (const QwtAbstractScaleDraw & *other*)

Copy constructor.

12.4.3.3 QwtAbstractScaleDraw::~~QwtAbstractScaleDraw () [virtual]

Destructor.

12.4.4 Member Function Documentation

12.4.4.1 void QwtAbstractScaleDraw::draw (QPainter * *painter*, const QPalette & *palette*) const [virtual]

Draw the scale.

Parameters

<i>painter</i>	The painter
<i>palette</i>	Palette, text color is used for the labels, foreground color for ticks and backbone

12.4.4.2 virtual void QwtAbstractScaleDraw::drawBackbone (QPainter * *painter*) const [protected, pure virtual]

Draws the baseline of the scale

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[drawTick\(\)](#), [drawLabel\(\)](#)

Implemented in [QwtRoundScaleDraw](#), and [QwtScaleDraw](#).

12.4.4.3 `virtual void QwtAbstractScaleDraw::drawLabel (QPainter * painter,
double value) const` `[protected, pure virtual]`

Draws the label for a major scale tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value

See also

[drawTick](#), [drawBackbone](#)

Implemented in [QwtRoundScaleDraw](#), and [QwtScaleDraw](#).

12.4.4.4 `virtual void QwtAbstractScaleDraw::drawTick (QPainter * painter,
double value, int len) const` `[protected, pure virtual]`

Draw a tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value of the tick
<i>len</i>	Lenght of the tick

See also

[drawBackbone\(\)](#), [drawLabel\(\)](#)

Implemented in [QwtRoundScaleDraw](#), and [QwtScaleDraw](#).

12.4.4.5 `void QwtAbstractScaleDraw::enableComponent (ScaleComponent
component, bool enable = true)`

En/Disable a component of the scale

Parameters

<i>component</i>	Scale component
<i>enable</i>	On/Off

See also

[hasComponent\(\)](#)

12.4.4.6 `virtual int QwtAbstractScaleDraw::extent (const QPen &, const QFont &) const [pure virtual]`

Calculate the extent

The extent is the distance from the baseline to the outermost pixel of the scale draw in opposite to its orientation. It is at least [minimumExtent\(\)](#) pixels.

See also

[setMinimumExtent\(\)](#), [minimumExtent\(\)](#)

Implemented in [QwtRoundScaleDraw](#), and [QwtScaleDraw](#).

12.4.4.7 `bool QwtAbstractScaleDraw::hasComponent (ScaleComponent component) const`

Check if a component is enabled

See also

[enableComponent\(\)](#)

12.4.4.8 `void QwtAbstractScaleDraw::invalidateCache () [protected]`

Invalidate the cache used by [QwtAbstractScaleDraw::tickLabel](#)

The cache is invalidated, when a new [QwtScaleDiv](#) is set. If the labels need to be changed. while the same [QwtScaleDiv](#) is set, [QwtAbstractScaleDraw::invalidateCache](#) needs to be called manually.

12.4.4.9 `QwtText QwtAbstractScaleDraw::label (double value) const [virtual]`

Convert a value into its representing label.

The value is converted to a plain text using `QLocale::system().toString(value)`. This method is often overloaded by applications to have individual labels.

Parameters

<i>value</i>	Value
--------------	-------

Returns

Label string.

Reimplemented in [QwtDialScaleDraw](#).

12.4.4.10 int QwtAbstractScaleDraw::majTickLength () const

The same as `QwtAbstractScaleDraw::tickLength(QwtScaleDiv::MajorTick)`.

12.4.4.11 const QwtScaleMap & QwtAbstractScaleDraw::map () const**Returns**

Map how to translate between scale and pixel values

12.4.4.12 int QwtAbstractScaleDraw::minimumExtent () const

Get the minimum extent

See also

[extent\(\)](#), [setMinimumExtent\(\)](#)

12.4.4.13 QwtAbstractScaleDraw & QwtAbstractScaleDraw::operator= (const QwtAbstractScaleDraw & *other*)

Assignment operator.

12.4.4.14 const QwtScaleDiv & QwtAbstractScaleDraw::scaleDiv () const**Returns**

scale division

12.4.4.15 QwtScaleMap & QwtAbstractScaleDraw::scaleMap ()**Returns**

Map how to translate between scale and pixel values

12.4.4.16 void QwtAbstractScaleDraw::setMinimumExtent (int *minExtent*)

Set a minimum for the extent.

The extent is calculated from the components of the scale draw. In situations, where the labels are changing and the layout depends on the extent (f.e scrolling a scale), setting an upper limit as minimum extent will avoid jumps of the layout.

Parameters

<i>minExtent</i>	Minimum extent
------------------	----------------

See also

[extent\(\)](#), [minimumExtent\(\)](#)

12.4.4.17 void QwtAbstractScaleDraw::setScaleDiv (const QwtScaleDiv & *sd*)

Change the scale division

Parameters

<i>sd</i>	New scale division
-----------	--------------------

12.4.4.18 void QwtAbstractScaleDraw::setSpacing (int *spacing*)

Set the spacing between tick and labels.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

[spacing\(\)](#)

12.4.4.19 void QwtAbstractScaleDraw::setTickLength (QwtScaleDiv::TickType *tickType*, int *length*)

Set the length of the ticks

Parameters

<i>tickType</i>	Tick type
<i>length</i>	New length

Warning

the length is limited to [0..1000]

**12.4.4.20 void QwtAbstractScaleDraw::setTransformation (
 QwtScaleTransformation * *transformation*)**

Change the transformation of the scale

Parameters

<i>transformation</i>	New scale transformation
-----------------------	--------------------------

12.4.4.21 int QwtAbstractScaleDraw::spacing () const

Get the spacing.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

See also

[setSpacing\(\)](#)

**12.4.4.22 const QwtText & QwtAbstractScaleDraw::tickLabel (const QFont &
 font, double *value*) const [protected]**

Convert a value into its representing label and cache it.

The conversion between value and label is called very often in the layout and painting code. Unfortunately the calculation of the label sizes might be slow (really slow for rich text in Qt4), so it's necessary to cache the labels.

Parameters

<i>font</i>	Font
<i>value</i>	Value

Returns

Tick label

12.4.4.23 `int QwtAbstractScaleDraw::tickLength (QwtScaleDiv::TickType
tickType) const`

Return the length of the ticks

See also

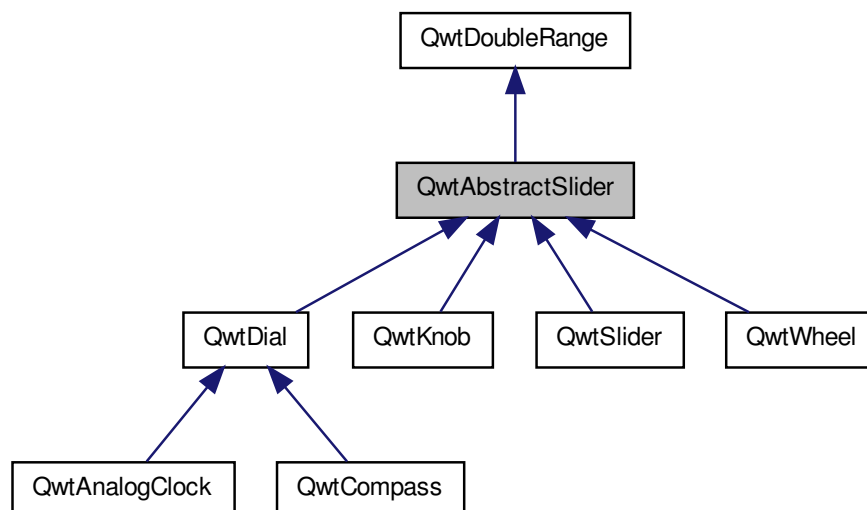
[setTickLength\(\)](#), [majTickLength\(\)](#)

12.5 QwtAbstractSlider Class Reference

An abstract base class for slider widgets.

```
#include <qwt_abstract_slider.h>
```

Inheritance diagram for QwtAbstractSlider:



Public Types

- enum [ScrollMode](#) {
 ScrNone,
 ScrMouse,
 ScrTimer,
 ScrDirect,

ScrPage }

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- virtual void [setReadOnly](#) (bool)
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()
- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- virtual void [incPages](#) (int)
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- virtual double [mass](#) () const
- double [maxValue](#) () const
- double [minValue](#) () const
- Qt::Orientation [orientation](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtAbstractSlider](#) (Qt::Orientation, QWidget *parent=NULL)
- virtual void [setMass](#) (double val)
- virtual void [setOrientation](#) (Qt::Orientation o)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setStep](#) (double)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)
- double [step](#) () const
- void [stopMoving](#) ()
- double [value](#) () const
- virtual [~QwtAbstractSlider](#) ()

Protected Member Functions

- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [getScrollMode](#) (const QPoint &p, int &scrollMode, int &direction)=0
- virtual double [getValue](#) (const QPoint &p)=0
- virtual void [keyPressEvent](#) (QKeyEvent *e)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)
- double [mouseOffset](#) () const
- virtual void [mousePressEvent](#) (QMouseEvent *e)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *e)
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- int [scrollMode](#) () const
- void [setMouseOffset](#) (double)
- virtual void [setPosition](#) (const QPoint &)
- virtual void [stepChange](#) ()
- virtual void [timerEvent](#) (QTimerEvent *e)
- virtual void [valueChange](#) ()
- virtual void [wheelEvent](#) (QWheelEvent *e)

12.5.1 Detailed Description

An abstract base class for slider widgets. [QwtAbstractSlider](#) is a base class for slider widgets. It handles mouse events and updates the slider's value accordingly. Derived classes only have to implement the [getValue\(\)](#) and [getScrollMode\(\)](#) members, and should react to a [valueChange\(\)](#), which normally requires repainting.

12.5.2 Member Enumeration Documentation**12.5.2.1 enum QwtAbstractSlider::ScrollMode**

Scroll mode

See also

[getScrollMode\(\)](#)

12.5.3 Constructor & Destructor Documentation**12.5.3.1 QwtAbstractSlider::QwtAbstractSlider (Qt::Orientation *orientation*, QWidget * *parent* = *NULL*) [explicit]**

Constructor.

Parameters

<i>orientation</i>	Orientation
<i>parent</i>	Parent widget

12.5.3.2 QwtAbstractSlider::~~QwtAbstractSlider () [virtual]

Destructor.

12.5.4 Member Function Documentation

12.5.4.1 double QwtDoubleRange::exactPrevValue () const [protected, inherited]

Returns the exact previous value.

12.5.4.2 double QwtDoubleRange::exactValue () const [protected, inherited]

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.5.4.3 void QwtAbstractSlider::fitValue (double *value*) [virtual, slot]

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.5.4.4 virtual void QwtAbstractSlider::getScrollMode (const QPoint & *p*, int & *scrollMode*, int & *direction*) [protected, pure virtual]

Determine what to do when the user presses a mouse button.

This function is abstract and has to be implemented by derived classes. It is called on a mousePress event. The derived class can determine what should happen next in dependence of the position where the mouse was pressed by returning scrolling mode and direction. [QwtAbstractSlider](#) knows the following modes:

QwtAbstractSlider::ScrNone Scrolling switched off. Don't change the value.

QwtAbstractSlider::ScrMouse Change the value while the user keeps the button pressed and moves the mouse.

QwtAbstractSlider::ScrTimer Automatic scrolling. Increment the value in the specified direction as long as the user keeps the button pressed.

QwtAbstractSlider::ScrPage Automatic scrolling. Same as ScrTimer, but increment by page size.

Parameters

<i>p</i>	point where the mouse was pressed
----------	-----------------------------------

Return values

<i>scrollMode</i>	The scrolling mode
<i>direction</i>	direction: 1, 0, or -1.

Implemented in [QwtDial](#), [QwtSlider](#), and [QwtWheel](#).

12.5.4.5 virtual double QwtAbstractSlider::getValue (const QPoint & *p*) [protected, pure virtual]

Determine the value corresponding to a specified point.

This is an abstract virtual function which is called when the user presses or releases a mouse button or moves the mouse. It has to be implemented by the derived class.

Parameters

<i>p</i>	point
----------	-------

Implemented in [QwtDial](#), [QwtSlider](#), and [QwtWheel](#).

12.5.4.6 void QwtDoubleRange::incPages (int *nPages*) [virtual, inherited]

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.5.4.7 void QwtAbstractSlider::incValue (int *steps*) [virtual, slot]

Increment the value by a specified number of steps.

Parameters

<i>steps</i>	number of steps
--------------	-----------------

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.5.4.8 bool QwtAbstractSlider::isReadOnly () const

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.5.4.9 bool QwtAbstractSlider::isValid () const [inline]

See also

[QwtDbIRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.5.4.10 void QwtAbstractSlider::keyPressEvent (QKeyEvent * *e*) [protected, virtual]

Handles key events

- Key_Down, Key_Left
Decrement by 1
- Key_Up, Key_Right
Increment by 1

Parameters

<i>e</i>	Key event
----------	-----------

See also

[isReadOnly\(\)](#)

Reimplemented in [QwtCompass](#), and [QwtDial](#).

12.5.4.11 double QwtAbstractSlider::mass () const [virtual]

Returns

mass

See also

[setMass\(\)](#)

Reimplemented in [QwtWheel](#).

12.5.4.12 double QwtDoubleRange::maxValue () const [inherited]

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.5.4.13 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

**12.5.4.14 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * *e*)
[protected, virtual]**

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

**12.5.4.15 void QwtAbstractSlider::mousePressEvent (QMouseEvent * *e*)
[protected, virtual]**

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

**12.5.4.16 void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * *e*)
[protected, virtual]**

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.5.4.17 Qt::Orientation QwtAbstractSlider::orientation () const

Returns

Orientation

See also

[setOrientation\(\)](#)

12.5.4.18 int QwtDoubleRange::pageSize () const [inherited]

Returns the page size in steps.

12.5.4.19 bool QwtDoubleRange::periodic () const [inherited]

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.5.4.20 double QwtDoubleRange::prevValue () const [protected, inherited]

Returns the previous value.

12.5.4.21 void QwtDoubleRange::rangeChange () [protected, virtual, inherited]

Notify a change of the range.

This virtual function is called whenever the range changes. The default implementation does nothing.

Reimplemented in [QwtCounter](#), [QwtDial](#), and [QwtSlider](#).

12.5.4.22 void QwtAbstractSlider::setMass (double *val*) [virtual]

Set the slider's mass for flywheel effect.

If the slider's mass is greater than 0, it will continue to move after the mouse button has been released. Its speed decreases with time at a rate depending on the slider's mass. A large mass means that it will continue to move for a long time.

Derived widgets may overload this function to make it public.

Parameters

<i>val</i>	New mass in kg
------------	----------------

See also[mass\(\)](#)Reimplemented in [QwtWheel](#).**12.5.4.23 void QwtAbstractSlider::setOrientation (Qt::Orientation *o*)
[virtual]**

Set the orientation.

Parameters

<i>o</i>	Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.
----------	--

Reimplemented in [QwtSlider](#), and [QwtWheel](#).**12.5.4.24 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]**

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

**12.5.4.25 void QwtAbstractSlider::setPosition (const QPoint & *p*)
[protected, virtual]**

Move the slider to a specified point, adjust the value and emit signals if necessary.

**12.5.4.26 void QwtDoubleRange::setRange (double *vmin*, double *vmax*,
double *vstep* = 0.0, int *pageSize* = 1) [inherited]**

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will **not** be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.5.4.27 void QwtAbstractSlider::setReadOnly (bool *readOnly*) [virtual, slot]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also

[isReadOnly\(\)](#)

12.5.4.28 void QwtDoubleRange::setStep (double *vstep*) [inherited]

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

12.5.4.29 void QwtAbstractSlider::setTracking (bool *enable*)

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i>	true (enable) or false (disable) tracking.
---------------	--

12.5.4.30 void QwtAbstractSlider::setUpdateTime (int *t*)

Specify the update interval for automatic scrolling.

Parameters

<i>t</i>	update interval in milliseconds
----------	---------------------------------

See also

[getScrollMode\(\)](#)

12.5.4.31 void QwtAbstractSlider::setValid (bool *valid*) [inline]**Parameters**

<i>valid</i>	true/false
--------------	------------

See also

[QwtDbtRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.5.4.32 void QwtAbstractSlider::setValue (double *val*) [virtual, slot]

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i> new value

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.5.4.33 void QwtAbstractSlider::sliderMoved (double *value*) [signal]

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i> new value

12.5.4.34 void QwtAbstractSlider::sliderPressed () [signal]

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.5.4.35 void QwtAbstractSlider::sliderReleased () [signal]

This signal is emitted when the user releases the movable part of the slider.

12.5.4.36 double QwtDoubleRange::step () const [inherited]

Returns

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.5.4.37 `void QwtDoubleRange::stepChange ()` [**protected**,
virtual, **inherited**]

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.5.4.38 `void QwtAbstractSlider::stopMoving ()`

Stop updating if automatic scrolling is active.

12.5.4.39 `void QwtAbstractSlider::timerEvent (QTimerEvent * e)`
[**protected**, **virtual**]

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.5.4.40 `double QwtDoubleRange::value () const` [**inherited**]

Returns the current value.

Reimplemented in [QwtCounter](#).

12.5.4.41 `void QwtAbstractSlider::valueChange ()` [**protected**,
virtual]

Notify change of value

This function can be reimplemented by derived classes in order to keep track of changes, i.e. repaint the widget. The default implementation emits a [valueChanged\(\)](#) signal if tracking is enabled.

Reimplemented from [QwtDoubleRange](#).

Reimplemented in [QwtDial](#), [QwtSlider](#), and [QwtWheel](#).

12.5.4.42 `void QwtAbstractSlider::valueChanged (double value)` [**signal**]

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.5.4.43 void QwtAbstractSlider::wheelEvent (QWheelEvent * *e*)
[protected, virtual]

Wheel Event handler

Parameters

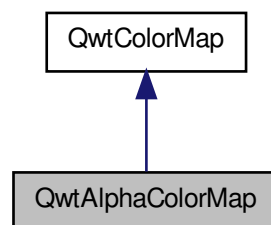
<i>e</i>	Wheel event
----------	-------------

12.6 QwtAlphaColorMap Class Reference

[QwtAlphaColorMap](#) varies the alpha value of a color.

```
#include <qwt_color_map.h>
```

Inheritance diagram for QwtAlphaColorMap:



Public Types

- enum [Format](#) {
 RGB,
 Indexed }

Public Member Functions

- QColor [color](#) () const

- QColor [color](#) (const [QwtDoubleInterval](#) &, double value) const
- virtual QVector< QRgb > [colorTable](#) (const [QwtDoubleInterval](#) &) const
- virtual [QwtColorMap](#) * [copy](#) () const
- [Format](#) [format](#) () const
- [QwtAlphaColorMap](#) & [operator=](#) (const [QwtAlphaColorMap](#) &)
- [QwtAlphaColorMap](#) (const QColor &=QColor(Qt::gray))
- [QwtAlphaColorMap](#) (const [QwtAlphaColorMap](#) &)
- virtual QRgb [rgb](#) (const [QwtDoubleInterval](#) &, double value) const
- void [setColor](#) (const QColor &)
- virtual [~QwtAlphaColorMap](#) ()

12.6.1 Detailed Description

[QwtAlphaColorMap](#) varies the alpha value of a color.

12.6.2 Member Enumeration Documentation

12.6.2.1 enum QwtColorMap::Format [inherited]

- RGB
The map is intended to map into QRgb values.
- Indexed
The map is intended to map into 8 bit values, that are indices into the color table.

See also

[rgb\(\)](#), [colorIndex\(\)](#), [colorTable\(\)](#)

12.6.3 Constructor & Destructor Documentation

12.6.3.1 QwtAlphaColorMap::QwtAlphaColorMap (const QColor & *color* = *QColor(Qt::gray)*)

Constructor

Parameters

<i>color</i>	Color of the map
--------------	------------------

12.6.3.2 QwtAlphaColorMap::QwtAlphaColorMap (const QwtAlphaColorMap & *other*)

Copy constructor

Parameters

<i>other</i>	Other color map
--------------	-----------------

12.6.3.3 QwtAlphaColorMap::~~QwtAlphaColorMap () [virtual]

Destructor.

12.6.4 Member Function Documentation**12.6.4.1 QColor QwtAlphaColorMap::color () const****Returns**

the color

See also

[setColor\(\)](#)

12.6.4.2 QColor QwtColorMap::color (const QwtDoubleInterval & *interval*, double *value*) const [inline, inherited]

Map a value into a color

Parameters

<i>interval</i>	Valid interval for values
<i>value</i>	Value

Returns

Color corresponding to value

Warning

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

12.6.4.3 QwtColorTable QwtColorMap::colorTable (const QwtDoubleInterval & *interval*) const [virtual, inherited]

Build and return a color map of 256 colors

The color table is needed for rendering indexed images in combination with using [colorIndex\(\)](#).

Parameters

<i>interval</i>	Range for the values
-----------------	----------------------

Returns

A color table, that can be used for a QImage

12.6.4.4 QwtColorMap * QwtAlphaColorMap::copy () const [virtual]

Clone the color map.

Implements [QwtColorMap](#).

12.6.4.5 QwtColorMap::Format QwtColorMap::format () const [inline, inherited]**Returns**

Intended format of the color map

See also

[Format](#)

12.6.4.6 QwtAlphaColorMap & QwtAlphaColorMap::operator= (const QwtAlphaColorMap & other)

Assignment operator

Parameters

<i>other</i>	Other color map
--------------	-----------------

Returns

*this

12.6.4.7 QRgb QwtAlphaColorMap::rgb (const QwtDoubleInterval & interval, double value) const [virtual]

Map a value of a given interval into a alpha value.

$\text{alpha} := (\text{value} - \text{interval.minValue}()) / \text{interval.width}();$

Parameters

<i>interval</i>	Range for all values
<i>value</i>	Value to map into a rgb value

Returns

rgb value, with an alpha value

Implements [QwtColorMap](#).

12.6.4.8 void QwtAlphaColorMap::setColor (const QColor & *color*)

Set the color

Parameters

<i>color</i>	Color
--------------	-------

See also

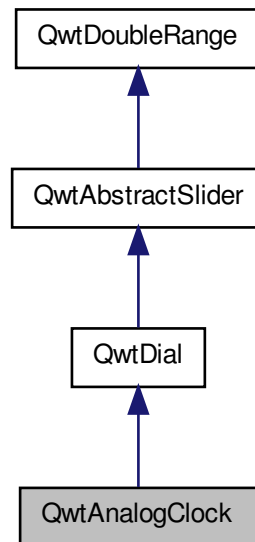
[color\(\)](#)

12.7 QwtAnalogClock Class Reference

An analog clock.

```
#include <qwt_analog_clock.h>
```

Inheritance diagram for QwtAnalogClock:



Public Types

- enum [Direction](#) {
 Clockwise,
 CounterClockwise }
- enum [Hand](#) {
 SecondHand,
 MinuteHand,
 HourHand,
 NHands }
- enum [Mode](#) {
 RotateNeedle,
 RotateScale }
- enum [ScaleOptions](#) {
 ScaleBackbone = 1,
 ScaleTicks = 2,
 ScaleLabel = 4 }

- enum [ScrollMode](#) {
 ScrNone,
 ScrMouse,
 ScrTimer,
 ScrDirect,
 ScrPage }
- enum [Shadow](#) {
 Plain = QFrame::Plain,
 Raised = QFrame::Raised,
 Sunken = QFrame::Sunken }

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- void [setCurrentTime](#) ()
- virtual void [setReadOnly](#) (bool)
- void [setTime](#) (const QTime &=QTime::currentTime())
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()
- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- QRect [boundingRect](#) () const
- QRect [contentsRect](#) () const
- [Direction](#) [direction](#) () const
- [Shadow](#) [frameShadow](#) () const
- [QwtDialNeedle](#) * [hand](#) ([Hand](#))
- const [QwtDialNeedle](#) * [hand](#) ([Hand](#)) const
- bool [hasVisibleBackground](#) () const
- virtual void [incPages](#) (int)
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- int [lineWidth](#) () const
- virtual double [mass](#) () const
- double [maxScaleArc](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const

- double [minScaleArc](#) () const
- double [minValue](#) () const
- [Mode](#) [mode](#) () const
- const [QwtDialNeedle](#) * [needle](#) () const
- [QwtDialNeedle](#) * [needle](#) ()
- Qt::Orientation [orientation](#) () const
- double [origin](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtAnalogClock](#) (QWidget *parent=NULL)
- virtual QRect [scaleContentsRect](#) () const
- [QwtDialScaleDraw](#) * [scaleDraw](#) ()
- const [QwtDialScaleDraw](#) * [scaleDraw](#) () const
- void [setDirection](#) ([Direction](#))
- void [setFrameShadow](#) ([Shadow](#))
- virtual void [setHand](#) ([Hand](#), [QwtDialNeedle](#) *)
- void [setLineWidth](#) (int)
- virtual void [setMass](#) (double val)
- void [setMode](#) ([Mode](#))
- virtual void [setOrientation](#) (Qt::Orientation o)
- virtual void [setOrigin](#) (double)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- virtual void [setScale](#) (int maxMajIntv, int maxMinIntv, double step=0.0)
- void [setScaleArc](#) (double min, double max)
- virtual void [setScaleDraw](#) ([QwtDialScaleDraw](#) *)
- void [setScaleOptions](#) (int)
- void [setScaleTicks](#) (int minLen, int medLen, int majLen, int penWidth=1)
- void [setStep](#) (double)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)
- virtual void [setWrapping](#) (bool)
- void [showBackground](#) (bool)
- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- void [stopMoving](#) ()
- double [value](#) () const
- bool [wrapping](#) () const
- virtual [~QwtAnalogClock](#) ()

Protected Member Functions

- virtual void [drawContents](#) (QPainter *) const
- virtual void [drawFocusIndicator](#) (QPainter *) const
- virtual void [drawFrame](#) (QPainter *p)
- virtual void [drawHand](#) (QPainter *, [Hand](#), const QPoint &, int radius, double direction, QPalette::ColorGroup) const
- virtual void [drawNeedle](#) (QPainter *, const QPoint &, int radius, double direction, QPalette::ColorGroup) const
- virtual void [drawScale](#) (QPainter *, const QPoint ¢er, int radius, double origin, double arcMin, double arcMax) const
- virtual void [drawScaleContents](#) (QPainter *painter, const QPoint ¢er, int radius) const
- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [getScrollMode](#) (const QPoint &, int &scrollMode, int &direction)
- virtual double [getValue](#) (const QPoint &)
- virtual void [keyPressEvent](#) (QKeyEvent *)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)
- double [mouseOffset](#) () const
- virtual void [mousePressEvent](#) (QMouseEvent *e)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *e)
- virtual void [paintEvent](#) (QPaintEvent *)
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- virtual void [resizeEvent](#) (QResizeEvent *)
- virtual [QwtText](#) [scaleLabel](#) (double) const
- int [scrollMode](#) () const
- void [setMouseOffset](#) (double)
- virtual void [setPosition](#) (const QPoint &)
- virtual void [stepChange](#) ()
- virtual void [timerEvent](#) (QTimerEvent *e)
- virtual void [updateMask](#) ()
- void [updateScale](#) ()
- virtual void [valueChange](#) ()
- virtual void [wheelEvent](#) (QWheelEvent *e)

12.7.1 Detailed Description

An analog clock.

Example

```
#include <qwt_analog_clock.h>

QwtAnalogClock *clock = new QwtAnalogClock(...);
clock->scaleDraw()->setPenWidth(3);
clock->setLineWidth(6);
```

```
clock->setFrameShadow(QwtDial::Sunken);
clock->setTime();

// update the clock every second
QTimer *timer = new QTimer(clock);
timer->connect(timer, SIGNAL(timeout()), clock, SLOT(setCurrentTime()));
timer->start(1000);
```

Qwt is missing a set of good looking hands. Contributions are very welcome.

Note

The examples/dials example shows how to use [QwtAnalogClock](#).

12.7.2 Member Enumeration Documentation

12.7.2.1 enum QwtDial::Direction [inherited]

Direction of the dial

12.7.2.2 enum QwtAnalogClock::Hand

Hand type

See also

[setHand\(\)](#), [hand\(\)](#)

12.7.2.3 enum QwtDial::Mode [inherited]

In case of RotateNeedle the needle is rotating, in case of RotateScale, the needle points to [origin\(\)](#) and the scale is rotating.

12.7.2.4 enum QwtDial::ScaleOptions [inherited]

see [QwtDial::setScaleOptions](#)

12.7.2.5 enum QwtAbstractSlider::ScrollMode [inherited]

Scroll mode

See also

[getScrollMode\(\)](#)

12.7.2.6 enum QwtDial::Shadow [inherited]

Frame shadow.

Unfortunately it is not possible to use QFrame::Shadow as a property of a widget that is not derived from QFrame. The following enum is made for the designer only. It is safe to use QFrame::Shadow instead.

12.7.3 Constructor & Destructor Documentation**12.7.3.1 QwtAnalogClock::QwtAnalogClock (QWidget * *parent* = *NULL*) [explicit]**

Constructor

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.7.3.2 QwtAnalogClock::~QwtAnalogClock () [virtual]

Destructor.

12.7.4 Member Function Documentation**12.7.4.1 QRect QwtDial::boundingRect () const [inherited]****Returns**

bounding rect of the dial including the frame

See also

[setLineWidth\(\)](#), [scaleContentsRect\(\)](#), [contentsRect\(\)](#)

12.7.4.2 QRect QwtDial::contentsRect () const [inherited]**Returns**

bounding rect of the circle inside the frame

See also

[setLineWidth\(\)](#), [scaleContentsRect\(\)](#), [boundingRect\(\)](#)

12.7.4.3 QwtDial::Direction QwtDial::direction () const [inherited]**Returns**

Direction of the dial

The default direction of a dial is QwtDial::Clockwise

See also

[setDirection\(\)](#)

12.7.4.4 void QwtDial::drawContents (QPainter * *painter*) const [protected, virtual, inherited]

Draw the contents inside the frame.

QColorGroup::Background is the background color outside of the frame. QColorGroup::Base is the background color inside the frame. QColorGroup::Foreground is the background color inside the scale.

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[boundingRect\(\)](#), [contentsRect\(\)](#), [scaleContentsRect\(\)](#), [QWidget::setPalette\(\)](#)

12.7.4.5 void QwtDial::drawFocusIndicator (QPainter * *painter*) const [protected, virtual, inherited]

Draw a dotted round circle, if !isReadOnly()

Parameters

<i>painter</i>	Painter
----------------	---------

12.7.4.6 void QwtDial::drawFrame (QPainter * *painter*) [protected, virtual, inherited]

Draw the frame around the dial

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[lineWidth\(\)](#), [frameShadow\(\)](#)

12.7.4.7 void QwtAnalogClock::drawHand (QPainter * *painter*, Hand *hd*, const QPoint & *center*, int *radius*, double *direction*, QPalette::ColorGroup *cg*) const [protected, virtual]

Draw a clock hand

Parameters

<i>painter</i>	Painter
<i>hd</i>	Specify the type of hand
<i>center</i>	Center of the clock
<i>radius</i>	Maximum length for the hands
<i>direction</i>	Direction of the hand in degrees, counter clockwise
<i>cg</i>	ColorGroup

12.7.4.8 void QwtAnalogClock::drawNeedle (QPainter * *painter*, const QPoint & *center*, int *radius*, double *direction*, QPalette::ColorGroup *cg*) const [protected, virtual]

Draw the needle.

A clock has no single needle but three hands instead. drawNeedle translates [value\(\)](#) into directions for the hands and calls [drawHand\(\)](#).

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the clock
<i>radius</i>	Maximum length for the hands
<i>direction</i>	Dummy, not used.
<i>cg</i>	ColorGroup

See also

[drawHand\(\)](#)

Reimplemented from [QwtDial](#).

12.7.4.9 `void QwtDial::drawScale (QPainter * painter, const QPoint & center, int radius, double origin, double minArc, double maxArc) const` `[protected, virtual, inherited]`

Draw the scale

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial
<i>radius</i>	Radius of the scale
<i>origin</i>	Origin of the scale
<i>minArc</i>	Minimum of the arc
<i>maxArc</i>	Minimum of the arc

See also

[QwtAbstractScaleDraw::setAngleRange\(\)](#)

12.7.4.10 `void QwtDial::drawScaleContents (QPainter * painter, const QPoint & center, int radius) const` `[protected, virtual, inherited]`

Draw the contents inside the scale

Paints nothing.

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the contents circle
<i>radius</i>	Radius of the contents circle

Reimplemented in [QwtCompass](#).

12.7.4.11 `double QwtDoubleRange::exactPrevValue () const` `[protected, inherited]`

Returns the exact previous value.

12.7.4.12 `double QwtDoubleRange::exactValue () const` `[protected, inherited]`

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.7.4.13 `void QwtAbstractSlider::fitValue (double value) [virtual, slot, inherited]`

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.7.4.14 `QwtDial::Shadow QwtDial::frameShadow () const [inherited]`

Returns

Frame shadow /sa [setFrameShadow\(\)](#), [lineWidth\(\)](#), [QFrame::frameShadow](#)

12.7.4.15 `void QwtDial::getScrollMode (const QPoint & pos, int & scrollMode, int & direction) [protected, virtual, inherited]`

See [QwtAbstractSlider::getScrollMode\(\)](#)

Parameters

<i>pos</i>	point where the mouse was pressed
------------	-----------------------------------

Return values

<i>scrollMode</i>	The scrolling mode
<i>direction</i>	direction: 1, 0, or -1.

See also

[QwtAbstractSlider::getScrollMode\(\)](#)

Implements [QwtAbstractSlider](#).

12.7.4.16 `double QwtDial::getValue (const QPoint & pos) [protected, virtual, inherited]`

Find the value for a given position

Parameters

<i>pos</i>	Position
------------	----------

Returns

Value

Implements [QwtAbstractSlider](#).

12.7.4.17 `QwtDialNeedle * QwtAnalogClock::hand (Hand hd)`**Returns**

Clock hand

Parameters

<i>hd</i>	Specifies the type of hand
-----------	----------------------------

See also

[setHand\(\)](#)

12.7.4.18 `const QwtDialNeedle * QwtAnalogClock::hand (Hand hd) const`**Returns**

Clock hand

Parameters

<i>hd</i>	Specifies the type of hand
-----------	----------------------------

See also

[setHand\(\)](#)

12.7.4.19 `bool QwtDial::hasVisibleBackground () const [inherited]`

true when the area outside of the frame is visible

See also

[showBackground\(\)](#), [setMask\(\)](#)

12.7.4.20 void QwtDoubleRange::incPages (int *nPages*) [virtual, inherited]

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.7.4.21 void QwtAbstractSlider::incValue (int *steps*) [virtual, slot, inherited]

Increment the value by a specified number of steps.

Parameters

<i>steps</i>	number of steps
--------------	-----------------

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.7.4.22 bool QwtAbstractSlider::isReadOnly () const [inherited]

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.7.4.23 `bool QwtAbstractSlider::isValid () const [inline, inherited]`

See also

`QwtDbfRange::isValid()`

Reimplemented from [QwtDoubleRange](#).

12.7.4.24 `void QwtDial::keyPressEvent (QKeyEvent * event) [protected, virtual, inherited]`

Handles key events

- Key_Down, KeyLeft
Decrement by 1
- Key_Prior
Decrement by [pageSize\(\)](#)
- Key_Home
Set the value to [minValue\(\)](#)
- Key_Up, KeyRight
Increment by 1
- Key_Next
Increment by [pageSize\(\)](#)
- Key_End
Set the value to [maxValue\(\)](#)

Parameters

<i>event</i> Key event

See also

[isReadOnly\(\)](#)

Reimplemented from [QwtAbstractSlider](#).

Reimplemented in [QwtCompass](#).

12.7.4.25 `int QwtDial::lineWidth () const [inherited]`

Returns

Line width of the frame

See also

[setLineWidth\(\)](#), [frameShadow\(\)](#), [lineWidth\(\)](#)

12.7.4.26 `double QwtAbstractSlider::mass () const` **[virtual, inherited]**

Returns

mass

See also

[setMass\(\)](#)

Reimplemented in [QwtWheel](#).

12.7.4.27 `double QwtDial::maxScaleArc () const` **[inherited]**

Returns

Upper limit of the scale arc

12.7.4.28 `double QwtDoubleRange::maxValue () const` **[inherited]**

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.7.4.29 `QSize QwtDial::minimumSizeHint () const` **[virtual, inherited]**

Return a minimum size hint.

Warning

The return value of [QwtDial::minimumSizeHint\(\)](#) depends on the font and the scale.

12.7.4.30 double QwtDial::minScaleArc () const [inherited]**Returns**

Lower limit of the scale arc

12.7.4.31 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.7.4.32 QwtDial::Mode QwtDial::mode () const [inherited]**Returns**

mode of the dial.

The value of the dial is indicated by the difference between the origin and the direction of the needle. In case of [QwtDial::RotateNeedle](#) the scale arc is fixed to the [origin\(\)](#) and the needle is rotating, in case of [QwtDial::RotateScale](#), the needle points to [origin\(\)](#) and the scale is rotating.

The default mode is [QwtDial::RotateNeedle](#).

See also

[setMode\(\)](#), [origin\(\)](#), [setScaleArc\(\)](#), [value\(\)](#)

**12.7.4.33 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.7.4.34 `void QwtAbstractSlider::mousePressEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.7.4.35 `void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.7.4.36 `const QwtDialNeedle * QwtDial::needle () const` [inherited]

Returns

needle

See also

[setNeedle\(\)](#)

12.7.4.37 `QwtDialNeedle * QwtDial::needle ()` [inherited]

Returns

needle

See also

[setNeedle\(\)](#)

12.7.4.38 `Qt::Orientation QwtAbstractSlider::orientation () const` **[inherited]**

Returns

Orientation

See also

[setOrientation\(\)](#)

12.7.4.39 `double QwtDial::origin () const` **[inherited]**

The origin is the angle where scale and needle is relative to.

Returns

Origin of the dial

See also

[setOrigin\(\)](#)

12.7.4.40 `int QwtDoubleRange::pageSize () const` **[inherited]**

Returns the page size in steps.

12.7.4.41 `void QwtDial::paintEvent (QPaintEvent * e)` **[protected, virtual, inherited]**

Paint the dial

Parameters

<i>e</i>	Paint event
----------	-------------

12.7.4.42 `bool QwtDoubleRange::periodic () const` **[inherited]**

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.7.4.43 `double QwtDoubleRange::prevValue () const` `[protected, inherited]`

Returns the previous value.

12.7.4.44 `void QwtDial::rangeChange ()` `[protected, virtual, inherited]`

[QwtDoubleRange](#) update hook.

Reimplemented from [QwtDoubleRange](#).

12.7.4.45 `void QwtDial::resizeEvent (QResizeEvent * e)` `[protected, virtual, inherited]`

Resize the dial widget

Parameters

<i>e</i>	Resize event
----------	--------------

12.7.4.46 `QRect QwtDial::scaleContentsRect () const` `[virtual, inherited]`

Returns

rect inside the scale

See also

[setLineWidth\(\)](#), [boundingRect\(\)](#), [contentsRect\(\)](#)

12.7.4.47 `QwtDialScaleDraw * QwtDial::scaleDraw ()` `[inherited]`

Return the scale draw.

12.7.4.48 `const QwtDialScaleDraw * QwtDial::scaleDraw () const` `[inherited]`

Return the scale draw.

12.7.4.49 QwtText QwtAnalogClock::scaleLabel (double *value*) const [protected, virtual]

Find the scale label for a given value

Parameters

<i>value</i>	Value
--------------	-------

Returns

Label

Reimplemented from [QwtDial](#).

12.7.4.50 void QwtAnalogClock::setCurrentTime () [slot]

Set the current time.

This is the same as [QwtAnalogClock::setTime\(\)](#), but Qt < 3.0 can't handle default parameters for slots.

12.7.4.51 void QwtDial::setDirection (Direction *direction*) [inherited]

Set the direction of the dial (clockwise/counterclockwise)

Direction *direction*

See also

[direction\(\)](#)

12.7.4.52 void QwtDial::setFrameShadow (Shadow *shadow*) [inherited]

Sets the frame shadow value from the frame style.

Parameters

<i>shadow</i>	Frame shadow
---------------	--------------

See also

[setLineWidth\(\)](#), [QFrame::setFrameShadow\(\)](#)

12.7.4.53 void QwtAnalogClock::setHand (Hand *hand*, QwtDialNeedle * *needle*) [virtual]

Set a clockhand

Parameters

<i>hand</i>	Specifies the type of hand
<i>needle</i>	Hand

See also

[hand\(\)](#)

12.7.4.54 void QwtDial::setLineWidth (int *lineWidth*) [inherited]

Sets the line width

Parameters

<i>lineWidth</i>	Line width
------------------	------------

See also

[setFrameShadow\(\)](#)

12.7.4.55 void QwtAbstractSlider::setMass (double *val*) [virtual, inherited]

Set the slider's mass for flywheel effect.

If the slider's mass is greater then 0, it will continue to move after the mouse button has been released. Its speed decreases with time at a rate depending on the slider's mass. A large mass means that it will continue to move for a long time.

Derived widgets may overload this function to make it public.

Parameters

<i>val</i>	New mass in kg
------------	----------------

See also

[mass\(\)](#)

Reimplemented in [QwtWheel](#).

12.7.4.56 void QwtDial::setMode (Mode *mode*) [inherited]

Change the mode of the meter.

Parameters

<i>mode</i>	New mode
-------------	----------

The value of the meter is indicated by the difference between north of the scale and the direction of the needle. In case of `QwtDial::RotateNeedle` north is pointing to the [origin\(\)](#) and the needle is rotating, in case of `QwtDial::RotateScale`, the needle points to [origin\(\)](#) and the scale is rotating.

The default mode is `QwtDial::RotateNeedle`.

See also

[mode\(\)](#), [setValue\(\)](#), [setOrigin\(\)](#)

12.7.4.57 `void QwtAbstractSlider::setOrientation (Qt::Orientation o)`
[virtual, inherited]

Set the orientation.

Parameters

<i>o</i>	Orientation. Allowed values are <code>Qt::Horizontal</code> and <code>Qt::Vertical</code> .
----------	---

Reimplemented in [QwtSlider](#), and [QwtWheel](#).

12.7.4.58 `void QwtDial::setOrigin (double origin)` [virtual, inherited]

Change the origin.

The origin is the angle where scale and needle is relative to.

Parameters

<i>origin</i>	New origin
---------------	------------

See also

[origin\(\)](#)

12.7.4.59 `void QwtDoubleRange::setPeriodic (bool tf)` [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.7.4.60 void QwtAbstractSlider::setPosition (const QPoint & p) [protected, virtual, inherited]

Move the slider to a specified point, adjust the value and emit signals if necessary.

12.7.4.61 void QwtDoubleRange::setRange (double vmin, double vmax, double vstep = 0.0, int pageSize = 1) [inherited]

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will *not* be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.7.4.62 void QwtAbstractSlider::setReadOnly (bool readOnly) [virtual, slot, inherited]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also

[isReadOnly\(\)](#)

12.7.4.63 void QwtDial::setScale (int *maxMajIntv*, int *maxMinIntv*, double *step* = 0.0) [virtual, inherited]

Change the intervals of the scale

See also

QwtAbstractScaleDraw::setScale()

12.7.4.64 void QwtDial::setScaleArc (double *minArc*, double *maxArc*) [inherited]

Change the arc of the scale

Parameters

<i>minArc</i>	Lower limit
<i>maxArc</i>	Upper limit

12.7.4.65 void QwtDial::setScaleDraw (QwtDialScaleDraw * *scaleDraw*) [virtual, inherited]

Set an individual scale draw

Parameters

<i>scaleDraw</i>	Scale draw
------------------	------------

Warning

The previous scale draw is deleted

12.7.4.66 void QwtDial::setScaleOptions (int *options*) [inherited]

A wrapper method for accessing the scale draw.

- options == 0
No visible scale: setScaleDraw(NULL)
- options & ScaleBackbone
En/disable the backbone of the scale.

- options & ScaleTicks
En/disable the ticks of the scale.
- options & ScaleLabel
En/disable scale labels

See also

[QwtAbstractScaleDraw::enableComponent\(\)](#)

12.7.4.67 void QwtDial::setScaleTicks (int *minLen*, int *medLen*, int *majLen*,
int *penWidth* = 1) [inherited]

Assign length and width of the ticks

Parameters

<i>minLen</i>	Length of the minor ticks
<i>medLen</i>	Length of the medium ticks
<i>majLen</i>	Length of the major ticks
<i>penWidth</i>	Width of the pen for all ticks

See also

[QwtAbstractScaleDraw::setTickLength\(\)](#), [QwtDialScaleDraw::setPenWidth\(\)](#)

12.7.4.68 void QwtDoubleRange::setStep (double *vstep*) [inherited]

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

12.7.4.69 void QwtAnalogClock::setTime (const QTime & *time* =
QTime::currentTime()) [slot]

Set a time

Parameters

<i>time</i>	Time to display
-------------	-----------------

**12.7.4.70 void QwtAbstractSlider::setTracking (bool *enable*)
[inherited]**

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i>	true (enable) or false (disable) tracking.
---------------	--

12.7.4.71 void QwtAbstractSlider::setUpdateTime (int *t*) [inherited]

Specify the update interval for automatic scrolling.

Parameters

<i>t</i>	update interval in milliseconds
----------	---------------------------------

See also

[getScrollMode\(\)](#)

**12.7.4.72 void QwtAbstractSlider::setValid (bool *valid*) [inline,
inherited]**
Parameters

<i>valid</i>	true/false
--------------	------------

See also

[QwtDbtRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.7.4.73 void QwtAbstractSlider::setValue (double *val*) [virtual, slot, inherited]

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i>	new value
------------	-----------

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.7.4.74 void QwtDial::setWrapping (bool *wrapping*) [virtual, inherited]

Sets whether it is possible to step the value from the highest value to the lowest value and vice versa to on.

Parameters

<i>wrapping</i>	en/disables wrapping
-----------------	----------------------

See also

[wrapping\(\)](#), [QwtDoubleRange::periodic\(\)](#)

Note

The meaning of wrapping is like the wrapping property of QSpinBox, but not like it is used in QDial.

12.7.4.75 void QwtDial::showBackground (bool *show*) [inherited]

Show/Hide the area outside of the frame

Parameters

<i>show</i>	Show if true, hide if false
-------------	-----------------------------

See also

[hasVisibleBackground\(\)](#), [setMask\(\)](#)

Warning

When [QwtDial](#) is a toplevel widget the window border might disappear too.

12.7.4.76 QSize QwtDial::sizeHint () const [virtual, inherited]**Returns**

Size hint

12.7.4.77 void QwtAbstractSlider::sliderMoved (double *value*) [signal, inherited]

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i>	new value
--------------	-----------

12.7.4.78 void QwtAbstractSlider::sliderPressed () [signal, inherited]

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.7.4.79 void QwtAbstractSlider::sliderReleased () [signal, inherited]

This signal is emitted when the user releases the movable part of the slider.

12.7.4.80 double QwtDoubleRange::step () const [inherited]**Returns**

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.7.4.81 `void QwtDoubleRange::stepChange () [protected, virtual, inherited]`

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.7.4.82 `void QwtAbstractSlider::stopMoving () [inherited]`

Stop updating if automatic scrolling is active.

12.7.4.83 `void QwtAbstractSlider::timerEvent (QTimerEvent * e) [protected, virtual, inherited]`

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.7.4.84 `void QwtDial::updateMask () [protected, virtual, inherited]`

Update the mask of the dial.

In case of "hasVisibleBackground() == false", the background is transparent by a mask.

See also

[showBackground\(\)](#), [hasVisibleBackground\(\)](#)

12.7.4.85 `void QwtDial::updateScale () [protected, inherited]`

Update the scale with the current attributes

See also

[setScale\(\)](#)

12.7.4.86 double QwtDoubleRange::value () const [inherited]

Returns the current value.

Reimplemented in [QwtCounter](#).

12.7.4.87 void QwtDial::valueChange () [protected, virtual, inherited]

[QwtDoubleRange](#) update hook.

Reimplemented from [QwtAbstractSlider](#).

12.7.4.88 void QwtAbstractSlider::valueChanged (double *value*) [signal, inherited]

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.7.4.89 void QwtAbstractSlider::wheelEvent (QWheelEvent * *e*) [protected, virtual, inherited]

Wheel Event handler

Parameters

<i>e</i>	Wheel event
----------	-------------

12.7.4.90 bool QwtDial::wrapping () const [inherited]

[wrapping\(\)](#) holds whether it is possible to step the value from the highest value to the lowest value and vice versa.

See also

[setWrapping\(\)](#), [QwtDoubleRange::setPeriodic\(\)](#)

Note

The meaning of wrapping is like the wrapping property of `QSpinBox`, but not like

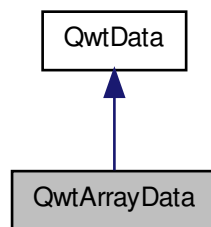
it is used in QDial.

12.8 QwtArrayData Class Reference

Data class containing two QwtArray<double> objects.

```
#include <qwt_data.h>
```

Inheritance diagram for QwtArrayData:



Public Member Functions

- virtual QwtDoubleRect [boundingRect](#) () const
- virtual [QwtData](#) * [copy](#) () const
- [QwtArrayData](#) & [operator=](#) (const [QwtArrayData](#) &)
- [QwtArrayData](#) (const double *x, const double *y, size_t size)
- [QwtArrayData](#) (const QwtArray< double > &x, const QwtArray< double > &y)
- virtual size_t [size](#) () const
- virtual double [x](#) (size_t i) const
- const QwtArray< double > & [xData](#) () const
- virtual double [y](#) (size_t i) const
- const QwtArray< double > & [yData](#) () const

12.8.1 Detailed Description

Data class containing two QwtArray<double> objects.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 QwtArrayData::QwtArrayData (const QwtArray< double > & *x*,
const QwtArray< double > & *y*)

Constructor

Parameters

<i>x</i>	Array of x values
<i>y</i>	Array of y values

See also

[QwtPlotCurve::setData\(\)](#)12.8.2.2 QwtArrayData::QwtArrayData (const double * *x*, const double * *y*,
size_t *size*)

Constructor

Parameters

<i>x</i>	Array of x values
<i>y</i>	Array of y values
<i>size</i>	Size of the x and y arrays

See also

[QwtPlotCurve::setData\(\)](#)

12.8.3 Member Function Documentation

12.8.3.1 QwtDoubleRect QwtArrayData::boundingRect () const
[virtual]

Returns the bounding rectangle of the data. If there is no bounding rect, like for empty data the rectangle is invalid: `QwtDoubleRect::isValid() == false`

Reimplemented from [QwtData](#).

12.8.3.2 QwtData * QwtArrayData::copy () const [virtual]

Returns

Pointer to a copy (virtual copy constructor)

Implements [QwtData](#).

12.8.3.3 QwtArrayData & QwtArrayData::operator= (const QwtArrayData & data)

Assignment.

12.8.3.4 size_t QwtArrayData::size () const [virtual]

Returns

Size of the data set

Implements [QwtData](#).

12.8.3.5 double QwtArrayData::x (size_t i) const [virtual]

Return the x value of data point i

Parameters

<i>i</i>	Index
----------	-------

Returns

x X value of data point i

Implements [QwtData](#).

12.8.3.6 const QwtArray< double > & QwtArrayData::xData () const

Returns

Array of the x-values

12.8.3.7 double QwtArrayData::y (size_t i) const [virtual]

Return the y value of data point i

Parameters

<i>i</i>	Index
----------	-------

Returns

y Y value of data point i

Implements [QwtData](#).

12.8.3.8 `const QwtArray< double > & QwtArrayData::yData () const`

Returns

Array of the y-values

12.9 QwtArrowButton Class Reference

Arrow Button.

```
#include <qwt_arrow_button.h>
```

Public Member Functions

- Qt::ArrowType [arrowType](#) () const
- virtual QSize [minimumSizeHint](#) () const
- int [num](#) () const
- [QwtArrowButton](#) (int num, Qt::ArrowType, QWidget *parent=NULL)
- virtual QSize [sizeHint](#) () const
- virtual [~QwtArrowButton](#) ()

Protected Member Functions

- virtual QSize [arrowSize](#) (Qt::ArrowType, const QSize &boundingSize) const
- virtual void [drawArrow](#) (QPainter *, const QRect &, Qt::ArrowType) const
- virtual void [drawButtonLabel](#) (QPainter *p)
- virtual void [keyPressEvent](#) (QKeyEvent *)
- virtual QRect [labelRect](#) () const
- virtual void [paintEvent](#) (QPaintEvent *event)

12.9.1 Detailed Description

Arrow Button. A push button with one or more filled triangles on its front. An Arrow button can have 1 to 3 arrows in a row, pointing up, down, left or right.

12.9.2 Constructor & Destructor Documentation

12.9.2.1 `QwtArrowButton::QwtArrowButton (int num, Qt::ArrowType arrowType, QWidget *parent = NULL) [explicit]`

Parameters

<i>num</i>	Number of arrows
<i>arrowType</i>	see Qt::ArowType in the Qt docs.
<i>parent</i>	Parent widget

12.9.2.2 QwtArrowButton::~QwtArrowButton () [virtual]

Destructor.

12.9.3 Member Function Documentation**12.9.3.1 QSize QwtArrowButton::arrowSize (Qt::ArrowType *arrowType*, const QSize & *boundingSize*) const [protected, virtual]**

Calculate the size for a arrow that fits into a rect of a given size

Parameters

<i>arrowType</i>	Arrow type
<i>bounding-Size</i>	Bounding size

Returns

Size of the arrow

12.9.3.2 Qt::ArrowType QwtArrowButton::arrowType () const

The direction of the arrows.

12.9.3.3 void QwtArrowButton::drawArrow (QPainter **painter*, const QRect & *r*, Qt::ArrowType *arrowType*) const [protected, virtual]

Draw an arrow int a bounding rect

Parameters

<i>painter</i>	Painter
<i>r</i>	Rectangle where to paint the arrow
<i>arrowType</i>	Arrow type

12.9.3.4 `void QwtArrowButton::drawButtonLabel (QPainter * painter)`
`[protected, virtual]`

Draw the button label.

Parameters

<i>painter</i>	Painter
----------------	---------

See also

The Qt Manual on QPushButton

12.9.3.5 `void QwtArrowButton::keyPressEvent (QKeyEvent * e)`
`[protected, virtual]`

autoRepeat for the space keys

12.9.3.6 `QRect QwtArrowButton::labelRect () const` `[protected, virtual]`

Returns

the bounding rect for the label

12.9.3.7 `QSize QwtArrowButton::minimumSizeHint () const` `[virtual]`

Return a minimum size hint.

12.9.3.8 `int QwtArrowButton::num () const`

The number of arrows.

12.9.3.9 `void QwtArrowButton::paintEvent (QPaintEvent * event)`
`[protected, virtual]`

Paint event handler

Parameters

<i>event</i>	Paint event
--------------	-------------

12.9.3.10 QSize QwtArrowButton::sizeHint () const [virtual]

Returns

a size hint

12.10 QwtClipper Class Reference

Some clipping algos.

```
#include <qwt_clipper.h>
```

Static Public Member Functions

- static QwtArray< [QwtDoubleInterval](#) > [clipCircle](#) (const QwtDoubleRect &, const QwtDoublePoint &, double radius)
- static QwtPolygon [clipPolygon](#) (const QRect &, const QwtPolygon &)
- static QwtPolygonF [clipPolygonF](#) (const QwtDoubleRect &, const QwtPolygonF &)

12.10.1 Detailed Description

Some clipping algos.

12.10.2 Member Function Documentation

12.10.2.1 QwtArray< QwtDoubleInterval > QwtClipper::clipCircle (const QwtDoubleRect & *clipRect*, const QwtDoublePoint & *center*, double *radius*) [static]

Circle clipping

[clipCircle\(\)](#) divides a circle into intervals of angles representing arcs of the circle. When the circle is completely inside the clip rectangle an interval [0.0, 2 * M_PI] is returned.

Parameters

<i>clipRect</i>	Clip rectangle
<i>center</i>	Center of the circle
<i>radius</i>	Radius of the circle

Returns

Arcs of the circle

**12.10.2.2 QwtPolygon QwtClipper::clipPolygon (const QRect & *clipRect*,
const QwtPolygon & *polygon*) [static]**

Sutherland-Hodgman polygon clipping

Parameters

<i>clipRect</i>	Clip rectangle
<i>polygon</i>	Polygon

Returns

Clipped polygon

**12.10.2.3 QwtPolygonF QwtClipper::clipPolygonF (const QwtDoubleRect &
clipRect, const QwtPolygonF & *polygon*) [static]**

Sutherland-Hodgman polygon clipping

Parameters

<i>clipRect</i>	Clip rectangle
<i>polygon</i>	Polygon

Returns

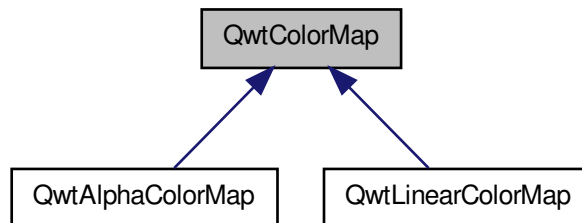
Clipped polygon

12.11 QwtColorMap Class Reference

[QwtColorMap](#) is used to map values into colors.

```
#include <qwt_color_map.h>
```

Inheritance diagram for QwtColorMap:



Public Types

- enum [Format](#) {
 RGB,
 Indexed }

Public Member Functions

- QColor [color](#) (const [QwtDoubleInterval](#) &, double value) const
- virtual unsigned char [colorIndex](#) (const [QwtDoubleInterval](#) &interval, double value) const =0
- virtual QVector< QRgb > [colorTable](#) (const [QwtDoubleInterval](#) &) const
- virtual [QwtColorMap](#) * [copy](#) () const =0
- [Format](#) [format](#) () const
- [QwtColorMap](#) ([Format](#)=[QwtColorMap::RGB](#))
- virtual QRgb [rgb](#) (const [QwtDoubleInterval](#) &interval, double value) const =0
- virtual [~QwtColorMap](#) ()

12.11.1 Detailed Description

[QwtColorMap](#) is used to map values into colors. For displaying 3D data on a 2D plane the 3rd dimension is often displayed using colors, like f.e in a spectrogram.

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

- QImage::Format_Indexed8
- QImage::Format_ARGB32

See also

[QwtPlotSpectrogram](#), [QwtScaleWidget](#)

12.11.2 Member Enumeration Documentation

12.11.2.1 enum QwtColorMap::Format

- RGB

The map is intended to map into QRgb values.

- Indexed

The map is intended to map into 8 bit values, that are indices into the color table.

See also

[rgb\(\)](#), [colorIndex\(\)](#), [colorTable\(\)](#)

12.11.3 Constructor & Destructor Documentation

12.11.3.1 QwtColorMap::QwtColorMap (Format *format* = *QwtColorMap::RGB*)

Constructor.

12.11.3.2 QwtColorMap::~QwtColorMap () [virtual]

Destructor.

12.11.4 Member Function Documentation

12.11.4.1 QColor QwtColorMap::color (const QwtDoubleInterval & *interval*, double *value*) const [inline]

Map a value into a color

Parameters

<i>interval</i>	Valid interval for values
<i>value</i>	Value

Returns

Color corresponding to value

Warning

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

12.11.4.2 `virtual unsigned char QwtColorMap::colorIndex (const QwtDoubleInterval & interval, double value) const` **[pure virtual]**

Map a value of a given interval into a color index

Parameters

<i>interval</i>	Range for the values
<i>value</i>	Value

Returns

color index, corresponding to value

Implemented in [QwtLinearColorMap](#).

12.11.4.3 `QwtColorTable QwtColorMap::colorTable (const QwtDoubleInterval & interval) const` **[virtual]**

Build and return a color map of 256 colors

The color table is needed for rendering indexed images in combination with using [colorIndex\(\)](#).

Parameters

<i>interval</i>	Range for the values
-----------------	----------------------

Returns

A color table, that can be used for a QImage

12.11.4.4 `virtual QwtColorMap* QwtColorMap::copy () const` **[pure virtual]**

Clone the color map.

Implemented in [QwtLinearColorMap](#), and [QwtAlphaColorMap](#).

12.11.4.5 QwtColorMap::Format QwtColorMap::format () const [inline]

Returns

Intended format of the color map

See also

[Format](#)

12.11.4.6 virtual QRgb QwtColorMap::rgb (const QwtDoubleInterval & *interval*, double *value*) const [pure virtual]

Map a value of a given interval into a rgb value.

Parameters

<i>interval</i>	Range for the values
<i>value</i>	Value

Returns

rgb value, corresponding to value

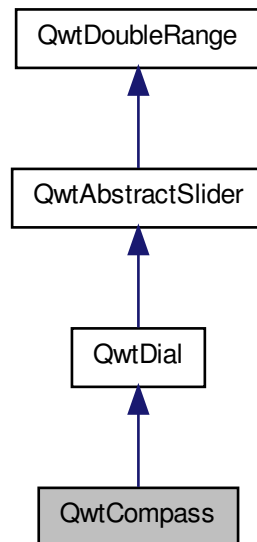
Implemented in [QwtLinearColorMap](#), and [QwtAlphaColorMap](#).

12.12 QwtCompass Class Reference

A Compass Widget.

```
#include <qwt_compass.h>
```

Inheritance diagram for QwtCompass:



Public Types

- enum [Direction](#) {
 Clockwise,
 CounterClockwise }
- enum [Mode](#) {
 RotateNeedle,
 RotateScale }
- enum [ScaleOptions](#) {
 ScaleBackbone = 1,
 ScaleTicks = 2,
 ScaleLabel = 4 }
- enum [ScrollMode](#) {
 ScrNone,
 ScrMouse,
 ScrTimer,
 ScrDirect,
 ScrPage }

- enum [Shadow](#) {
Plain = QFrame::Plain,
Raised = QFrame::Raised,
Sunken = QFrame::Sunken }

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- virtual void [setReadOnly](#) (bool)
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()
- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- QRect [boundingRect](#) () const
- QRect [contentsRect](#) () const
- [Direction](#) [direction](#) () const
- [Shadow](#) [frameShadow](#) () const
- bool [hasVisibleBackground](#) () const
- virtual void [incPages](#) (int)
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- const QMap< double, QString > & [labelMap](#) () const
- QMap< double, QString > & [labelMap](#) ()
- int [lineWidth](#) () const
- virtual double [mass](#) () const
- double [maxScaleArc](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const
- double [minScaleArc](#) () const
- double [minValue](#) () const
- [Mode](#) [mode](#) () const
- const QwtDialNeedle * [needle](#) () const
- QwtDialNeedle * [needle](#) ()
- Qt::Orientation [orientation](#) () const
- double [origin](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const

- [QwtCompass](#) (QWidget *parent=NULL)
- [QwtCompassRose](#) * [rose](#) ()
- const [QwtCompassRose](#) * [rose](#) () const
- virtual QRect [scaleContentsRect](#) () const
- [QwtDialScaleDraw](#) * [scaleDraw](#) ()
- const [QwtDialScaleDraw](#) * [scaleDraw](#) () const
- void [setDirection](#) (Direction)
- void [setFrameShadow](#) (Shadow)
- void [setLabelMap](#) (const QMap< double, QString > &map)
- void [setLineWidth](#) (int)
- virtual void [setMass](#) (double val)
- void [setMode](#) (Mode)
- virtual void [setNeedle](#) (QwtDialNeedle *)
- virtual void [setOrientation](#) (Qt::Orientation o)
- virtual void [setOrigin](#) (double)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setRose](#) (QwtCompassRose *rose)
- virtual void [setScale](#) (int maxMajIntv, int maxMinIntv, double step=0.0)
- void [setScaleArc](#) (double min, double max)
- virtual void [setScaleDraw](#) (QwtDialScaleDraw *)
- void [setScaleOptions](#) (int)
- void [setScaleTicks](#) (int minLen, int medLen, int majLen, int penWidth=1)
- void [setStep](#) (double)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)
- virtual void [setWrapping](#) (bool)
- void [showBackground](#) (bool)
- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- void [stopMoving](#) ()
- double [value](#) () const
- bool [wrapping](#) () const
- virtual ~[QwtCompass](#) ()

Protected Member Functions

- virtual void [drawContents](#) (QPainter *) const
- virtual void [drawFocusIndicator](#) (QPainter *) const
- virtual void [drawFrame](#) (QPainter *p)
- virtual void [drawNeedle](#) (QPainter *, const QPoint &, int radius, double direction, QPalette::ColorGroup) const
- virtual void [drawRose](#) (QPainter *, const QPoint ¢er, int radius, double north, QPalette::ColorGroup) const
- virtual void [drawScale](#) (QPainter *, const QPoint ¢er, int radius, double origin, double arcMin, double arcMax) const

- virtual void [drawScaleContents](#) (QPainter *, const QPoint ¢er, int radius) const
- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [getScrollMode](#) (const QPoint &, int &scrollMode, int &direction)
- virtual double [getValue](#) (const QPoint &)
- virtual void [keyPressEvent](#) (QKeyEvent *)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)
- double [mouseOffset](#) () const
- virtual void [mousePressEvent](#) (QMouseEvent *e)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *e)
- virtual void [paintEvent](#) (QPaintEvent *)
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- virtual void [resizeEvent](#) (QResizeEvent *)
- virtual [QwtText](#) [scaleLabel](#) (double value) const
- int [scrollMode](#) () const
- void [setMouseOffset](#) (double)
- virtual void [setPosition](#) (const QPoint &)
- virtual void [stepChange](#) ()
- virtual void [timerEvent](#) (QTimerEvent *e)
- virtual void [updateMask](#) ()
- void [updateScale](#) ()
- virtual void [valueChange](#) ()
- virtual void [wheelEvent](#) (QWheelEvent *e)

12.12.1 Detailed Description

A Compass Widget. [QwtCompass](#) is a widget to display and enter directions. It consists of a scale, an optional needle and rose.

Note

The examples/dials example shows how to use [QwtCompass](#).

12.12.2 Member Enumeration Documentation

12.12.2.1 enum QwtDial::Direction [\[inherited\]](#)

Direction of the dial

12.12.2.2 enum QwtDial::Mode [\[inherited\]](#)

In case of RotateNeedle the needle is rotating, in case of RotateScale, the needle points to [origin\(\)](#) and the scale is rotating.

12.12.2.3 enum QwtDial::ScaleOptions [inherited]

see [QwtDial::setScaleOptions](#)

12.12.2.4 enum QwtAbstractSlider::ScrollMode [inherited]

Scroll mode

See also

[getScrollMode\(\)](#)

12.12.2.5 enum QwtDial::Shadow [inherited]

Frame shadow.

Unfortunately it is not possible to use QFrame::Shadow as a property of a widget that is not derived from QFrame. The following enum is made for the designer only. It is safe to use QFrame::Shadow instead.

12.12.3 Constructor & Destructor Documentation**12.12.3.1 QwtCompass::QwtCompass (QWidget * *parent* = *NULL*) [explicit]**

Constructor.

Parameters

<i>parent</i> Parent widget

Create a compass widget with a scale, no needle and no rose. The default origin is 270.0 with no valid value. It accepts mouse and keyboard inputs and has no step size. The default mode is QwtDial::RotateNeedle.

12.12.3.2 QwtCompass::~QwtCompass () [virtual]

Destructor.

12.12.4 Member Function Documentation

12.12.4.1 QRect QwtDial::boundingRect () const [inherited]

Returns

bounding rect of the dial including the frame

See also

[setLineWidth\(\)](#), [scaleContentsRect\(\)](#), [contentsRect\(\)](#)

12.12.4.2 QRect QwtDial::contentsRect () const [inherited]

Returns

bounding rect of the circle inside the frame

See also

[setLineWidth\(\)](#), [scaleContentsRect\(\)](#), [boundingRect\(\)](#)

12.12.4.3 QwtDial::Direction QwtDial::direction () const [inherited]

Returns

Direction of the dial

The default direction of a dial is QwtDial::Clockwise

See also

[setDirection\(\)](#)

12.12.4.4 void QwtDial::drawContents (QPainter * *painter*) const [protected, virtual, inherited]

Draw the contents inside the frame.

QColorGroup::Background is the background color outside of the frame. QColorGroup::Base is the background color inside the frame. QColorGroup::Foreground is the background color inside the scale.

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[boundingRect\(\)](#), [contentsRect\(\)](#), [scaleContentsRect\(\)](#), [QWidget::setPalette\(\)](#)

12.12.4.5 void QwtDial::drawFocusIndicator (QPainter * *painter*) const
[protected, virtual, inherited]

Draw a dotted round circle, if !isReadOnly()

Parameters

<i>painter</i>	Painter
----------------	---------

12.12.4.6 void QwtDial::drawFrame (QPainter * *painter*) [protected,
virtual, inherited]

Draw the frame around the dial

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[lineWidth\(\)](#), [frameShadow\(\)](#)

12.12.4.7 void QwtDial::drawNeedle (QPainter * *painter*, const QPoint &
***center*, int *radius*, double *direction*, QPalette::ColorGroup *cg*) const**
[protected, virtual, inherited]

Draw the needle

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial
<i>radius</i>	Length for the needle
<i>direction</i>	Direction of the needle in degrees, counter clockwise
<i>cg</i>	ColorGroup

Reimplemented in [QwtAnalogClock](#).

12.12.4.8 `void QwtCompass::drawRose (QPainter * painter, const QPoint & center, int radius, double north, QPalette::ColorGroup cg) const` `[protected, virtual]`

Draw the compass rose

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the compass
<i>radius</i>	of the circle, where to paint the rose
<i>north</i>	Direction pointing north, in degrees counter clockwise
<i>cg</i>	Color group

12.12.4.9 `void QwtDial::drawScale (QPainter * painter, const QPoint & center, int radius, double origin, double minArc, double maxArc) const` `[protected, virtual, inherited]`

Draw the scale

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial
<i>radius</i>	Radius of the scale
<i>origin</i>	Origin of the scale
<i>minArc</i>	Minimum of the arc
<i>maxArc</i>	Minimum of the arc

See also

`QwtAbstractScaleDraw::setAngleRange()`

12.12.4.10 `void QwtCompass::drawScaleContents (QPainter * painter, const QPoint & center, int radius) const` `[protected, virtual]`

Draw the contents of the scale

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the content circle
<i>radius</i>	Radius of the content circle

Reimplemented from [QwtDial](#).

12.12.4.11 `double QwtDoubleRange::exactPrevValue () const`
`[protected, inherited]`

Returns the exact previous value.

12.12.4.12 `double QwtDoubleRange::exactValue () const` `[protected, inherited]`

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.12.4.13 `void QwtAbstractSlider::fitValue (double value)` `[virtual, slot, inherited]`

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.12.4.14 `QwtDial::Shadow QwtDial::frameShadow () const`
`[inherited]`

Returns

Frame shadow /sa [setFrameShadow\(\)](#), [lineWidth\(\)](#), [QFrame::frameShadow](#)

12.12.4.15 `void QwtDial::getScrollMode (const QPoint & pos, int & scrollMode, int & direction)` `[protected, virtual, inherited]`

See [QwtAbstractSlider::getScrollMode\(\)](#)

Parameters

<i>pos</i>	point where the mouse was pressed
------------	-----------------------------------

Return values

<i>scrollMode</i>	The scrolling mode
<i>direction</i>	direction: 1, 0, or -1.

See also

[QwtAbstractSlider::getScrollMode\(\)](#)

Implements [QwtAbstractSlider](#).

12.12.4.16 `double QwtDial::getValue (const QPoint & pos) [protected, virtual, inherited]`

Find the value for a given position

Parameters

<i>pos</i>	Position
------------	----------

Returns

Value

Implements [QwtAbstractSlider](#).

12.12.4.17 `bool QwtDial::hasVisibleBackground () const [inherited]`

true when the area outside of the frame is visible

See also

[showBackground\(\)](#), [setMask\(\)](#)

12.12.4.18 `void QwtDoubleRange::incPages (int nPages) [virtual, inherited]`

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.12.4.19 `void QwtAbstractSlider::incValue (int steps) [virtual, slot, inherited]`

Increment the value by a specified number of steps.

Parameters

<i>steps</i>	number of steps
--------------	-----------------

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.12.4.20 `bool QwtAbstractSlider::isReadOnly () const [inherited]`

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.12.4.21 `bool QwtAbstractSlider::isValid () const [inline, inherited]`

See also

[QwtDbIRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.12.4.22 `void QwtCompass::keyPressEvent (QKeyEvent * kev) [protected, virtual]`

Handles key events

Beside the keys described in [QwtDial::keyPressEvent](#) numbers from 1-9 (without 5) set the direction according to their position on the num pad.

See also

[isReadOnly\(\)](#)

Reimplemented from [QwtDial](#).

12.12.4.23 `const QMap< double, QString > & QwtCompass::labelMap ()`
`const`

Returns

map, mapping values to labels

See also

[setLabelMap\(\)](#)

12.12.4.24 `QMap< double, QString > & QwtCompass::labelMap ()`

Returns

map, mapping values to labels

See also

[setLabelMap\(\)](#)

12.12.4.25 `int QwtDial::lineWidth () const` `[inherited]`

Returns

Line width of the frame

See also

[setLineWidth\(\)](#), [frameShadow\(\)](#), [lineWidth\(\)](#)

12.12.4.26 `double QwtAbstractSlider::mass () const` `[virtual, inherited]`

Returns

mass

See also

[setMass\(\)](#)

Reimplemented in [QwtWheel](#).

12.12.4.27 double QwtDial::maxScaleArc () const [inherited]**Returns**

Upper limit of the scale arc

12.12.4.28 double QwtDoubleRange::maxValue () const [inherited]

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.12.4.29 QSize QwtDial::minimumSizeHint () const [virtual, inherited]

Return a minimum size hint.

Warning

The return value of [QwtDial::minimumSizeHint\(\)](#) depends on the font and the scale.

12.12.4.30 double QwtDial::minScaleArc () const [inherited]**Returns**

Lower limit of the scale arc

12.12.4.31 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.12.4.32 QwtDial::Mode QwtDial::mode () const [inherited]

Returns

mode of the dial.

The value of the dial is indicated by the difference between the origin and the direction of the needle. In case of QwtDial::RotateNeedle the scale arc is fixed to the [origin\(\)](#) and the needle is rotating, in case of QwtDial::RotateScale, the needle points to [origin\(\)](#) and the scale is rotating.

The default mode is QwtDial::RotateNeedle.

See also

[setMode\(\)](#), [origin\(\)](#), [setScaleArc\(\)](#), [value\(\)](#)

**12.12.4.33 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

**12.12.4.34 void QwtAbstractSlider::mousePressEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.12.4.35 `void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.12.4.36 `QwtDialNeedle * QwtDial::needle ()` [inherited]

Returns

needle

See also

[setNeedle\(\)](#)

12.12.4.37 `const QwtDialNeedle * QwtDial::needle () const` [inherited]

Returns

needle

See also

[setNeedle\(\)](#)

12.12.4.38 `Qt::Orientation QwtAbstractSlider::orientation () const`
[inherited]

Returns

Orientation

See also

[setOrientation\(\)](#)

12.12.4.39 double QwtDial::origin () const [inherited]

The origin is the angle where scale and needle is relative to.

Returns

Origin of the dial

See also

[setOrigin\(\)](#)

12.12.4.40 int QwtDoubleRange::pageSize () const [inherited]

Returns the page size in steps.

12.12.4.41 void QwtDial::paintEvent (QPaintEvent * e) [protected, virtual, inherited]

Paint the dial

Parameters

<i>e</i>	Paint event
----------	-------------

12.12.4.42 bool QwtDoubleRange::periodic () const [inherited]

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.12.4.43 double QwtDoubleRange::prevValue () const [protected, inherited]

Returns the previous value.

12.12.4.44 void QwtDial::rangeChange () [protected, virtual, inherited]

[QwtDoubleRange](#) update hook.

Reimplemented from [QwtDoubleRange](#).

12.12.4.45 `void QwtDial::resizeEvent (QResizeEvent * e)` `[protected, virtual, inherited]`

Resize the dial widget

Parameters

<i>e</i>	Resize event
----------	--------------

12.12.4.46 `const QwtCompassRose * QwtCompass::rose () const`

Returns

rose

See also

[setRose\(\)](#)

12.12.4.47 `QwtCompassRose * QwtCompass::rose ()`

Returns

rose

See also

[setRose\(\)](#)

12.12.4.48 `QRect QwtDial::scaleContentsRect () const` `[virtual, inherited]`

Returns

rect inside the scale

See also

[setLineWidth\(\)](#), [boundingRect\(\)](#), [contentsRect\(\)](#)

12.12.4.49 QwtDialScaleDraw * QwtDial::scaleDraw () [inherited]

Return the scale draw.

12.12.4.50 const QwtDialScaleDraw * QwtDial::scaleDraw () const [inherited]

Return the scale draw.

12.12.4.51 QwtText QwtCompass::scaleLabel (double *value*) const [protected, virtual]

Map a value to a corresponding label

Parameters

<i>value</i>	Value that will be mapped
--------------	---------------------------

Returns

Label, or QString::null

label() looks in a map for a corresponding label for value or return an null text.

See also

[labelMap\(\)](#), [setLabelMap\(\)](#)

Reimplemented from [QwtDial](#).

12.12.4.52 void QwtDial::setDirection (Direction *direction*) [inherited]

Set the direction of the dial (clockwise/counterclockwise)

Direction *direction*

See also

[direction\(\)](#)

12.12.4.53 void QwtDial::setFrameShadow (Shadow *shadow*) [inherited]

Sets the frame shadow value from the frame style.

Parameters

<i>shadow</i>	Frame shadow
---------------	--------------

See also

[setLineWidth\(\)](#), [QFrame::setFrameShadow\(\)](#)

12.12.4.54 void QwtCompass::setLabelMap (const QMap< double, QString > &map)

Set a map, mapping values to labels.

Parameters

<i>map</i>	value to label map
------------	--------------------

The values of the major ticks are found by looking into this map. The default map consists of the labels N, NE, E, SE, S, SW, W, NW.

Warning

The map will have no effect for values that are no major tick values. Major ticks can be changed by [QwtScaleDraw::setScale](#)

See also

[labelMap\(\)](#), [scaleDraw\(\)](#), [setScale\(\)](#)

12.12.4.55 void QwtDial::setLineWidth (int *lineWidth*) [inherited]

Sets the line width

Parameters

<i>lineWidth</i>	Line width
------------------	------------

See also

[setFrameShadow\(\)](#)

12.12.4.56 void QwtAbstractSlider::setMass (double *val*) [virtual, inherited]

Set the slider's mass for flywheel effect.

If the slider's mass is greater than 0, it will continue to move after the mouse button has been released. Its speed decreases with time at a rate depending on the slider's mass. A large mass means that it will continue to move for a long time.

Derived widgets may overload this function to make it public.

Parameters

<i>val</i>	New mass in kg
------------	----------------

See also

[mass\(\)](#)

Reimplemented in [QwtWheel](#).

12.12.4.57 void QwtDial::setMode (Mode mode) [inherited]

Change the mode of the meter.

Parameters

<i>mode</i>	New mode
-------------	----------

The value of the meter is indicated by the difference between north of the scale and the direction of the needle. In case of `QwtDial::RotateNeedle` north is pointing to the [origin\(\)](#) and the needle is rotating, in case of `QwtDial::RotateScale`, the needle points to [origin\(\)](#) and the scale is rotating.

The default mode is `QwtDial::RotateNeedle`.

See also

[mode\(\)](#), [setValue\(\)](#), [setOrigin\(\)](#)

12.12.4.58 void QwtDial::setNeedle (QwtDialNeedle * needle) [virtual, inherited]

Set a needle for the dial

Qwt is missing a set of good looking needles. Contributions are very welcome.

Parameters

<i>needle</i>	Needle
---------------	--------

Warning

The needle will be deleted, when a different needle is set or in [~QwtDial\(\)](#)

12.12.4.59 void QwtAbstractSlider::setOrientation (Qt::Orientation *o*) [virtual, inherited]

Set the orientation.

Parameters

<i>o</i>	Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.
----------	--

Reimplemented in [QwtSlider](#), and [QwtWheel](#).

12.12.4.60 void QwtDial::setOrigin (double *origin*) [virtual, inherited]

Change the origin.

The origin is the angle where scale and needle is relative to.

Parameters

<i>origin</i>	New origin
---------------	------------

See also

[origin\(\)](#)

12.12.4.61 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.12.4.62 `void QwtAbstractSlider::setPosition (const QPoint & p)`
[protected, virtual, inherited]

Move the slider to a specified point, adjust the value and emit signals if necessary.

12.12.4.63 `void QwtDoubleRange::setRange (double vmin, double vmax,
double vstep = 0.0, int pageSize = 1)` **[inherited]**

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will **not** be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.12.4.64 `void QwtAbstractSlider::setReadOnly (bool readOnly)`
[virtual, slot, inherited]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also

[isReadOnly\(\)](#)

12.12.4.65 `void QwtCompass::setRose (QwtCompassRose * rose)`

Set a rose for the compass

Parameters

<i>rose</i>	Compass rose
-------------	--------------

Warning

The rose will be deleted, when a different rose is set or in `~QwtCompass`

See also

[rose\(\)](#)

12.12.4.66 `void QwtDial::setScale (int maxMajIntv, int maxMinIntv, double step = 0.0) [virtual, inherited]`

Change the intervals of the scale

See also

`QwtAbstractScaleDraw::setScale()`

12.12.4.67 `void QwtDial::setScaleArc (double minArc, double maxArc) [inherited]`

Change the arc of the scale

Parameters

<i>minArc</i>	Lower limit
<i>maxArc</i>	Upper limit

12.12.4.68 `void QwtDial::setScaleDraw (QwtDialScaleDraw * scaleDraw) [virtual, inherited]`

Set an individual scale draw

Parameters

<i>scaleDraw</i>	Scale draw
------------------	------------

Warning

The previous scale draw is deleted

12.12.4.69 `void QwtDial::setScaleOptions (int options) [inherited]`

A wrapper method for accessing the scale draw.

- options == 0
No visible scale: `setScaleDraw(NULL)`
- options & `ScaleBackbone`
En/disable the backbone of the scale.
- options & `ScaleTicks`
En/disable the ticks of the scale.
- options & `ScaleLabel`
En/disable scale labels

See also

[QwtAbstractScaleDraw::enableComponent\(\)](#)

12.12.4.70 `void QwtDial::setScaleTicks (int minLen, int medLen, int majLen, int penWidth = 1) [inherited]`

Assign length and width of the ticks

Parameters

<i>minLen</i>	Length of the minor ticks
<i>medLen</i>	Length of the medium ticks
<i>majLen</i>	Length of the major ticks
<i>penWidth</i>	Width of the pen for all ticks

See also

[QwtAbstractScaleDraw::setTickLength\(\)](#), [QwtDialScaleDraw::setPenWidth\(\)](#)

12.12.4.71 `void QwtDoubleRange::setStep (double vstep) [inherited]`

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

**12.12.4.72 void QwtAbstractSlider::setTracking (bool *enable*)
[*inherited*]**

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i> true (enable) or false (disable) tracking.
--

12.12.4.73 void QwtAbstractSlider::setUpdateTime (int *t*) [*inherited*]

Specify the update interval for automatic scrolling.

Parameters

<i>t</i> update interval in milliseconds
--

See also

[getScrollMode\(\)](#)

**12.12.4.74 void QwtAbstractSlider::setValid (bool *valid*) [*inline*,
inherited]****Parameters**

<i>valid</i> true/false

See also

[QwtDbIRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.12.4.75 void QwtAbstractSlider::setValue (double *val*) [virtual, slot, inherited]

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i> new value

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.12.4.76 void QwtDial::setWrapping (bool *wrapping*) [virtual, inherited]

Sets whether it is possible to step the value from the highest value to the lowest value and vice versa to on.

Parameters

<i>wrapping</i> en/disables wrapping

See also

[wrapping\(\)](#), [QwtDoubleRange::periodic\(\)](#)

Note

The meaning of wrapping is like the wrapping property of QSpinBox, but not like it is used in QDial.

12.12.4.77 void QwtDial::showBackground (bool *show*) [inherited]

Show/Hide the area outside of the frame

Parameters

<i>show</i> Show if true, hide if false

See also

[hasVisibleBackground\(\)](#), [setMask\(\)](#)

Warning

When [QwtDial](#) is a toplevel widget the window border might disappear too.

12.12.4.78 QSize QwtDial::sizeHint () const [virtual, inherited]**Returns**

Size hint

12.12.4.79 void QwtAbstractSlider::sliderMoved (double *value*) [signal, inherited]

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i>	new value
--------------	-----------

12.12.4.80 void QwtAbstractSlider::sliderPressed () [signal, inherited]

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.12.4.81 void QwtAbstractSlider::sliderReleased () [signal, inherited]

This signal is emitted when the user releases the movable part of the slider.

12.12.4.82 double QwtDoubleRange::step () const [inherited]**Returns**

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.12.4.83 `void QwtDoubleRange::stepChange () [protected, virtual, inherited]`

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.12.4.84 `void QwtAbstractSlider::stopMoving () [inherited]`

Stop updating if automatic scrolling is active.

12.12.4.85 `void QwtAbstractSlider::timerEvent (QTimerEvent * e) [protected, virtual, inherited]`

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.12.4.86 `void QwtDial::updateMask () [protected, virtual, inherited]`

Update the mask of the dial.

In case of "hasVisibleBackground() == false", the background is transparent by a mask.

See also

[showBackground\(\)](#), [hasVisibleBackground\(\)](#)

12.12.4.87 `void QwtDial::updateScale () [protected, inherited]`

Update the scale with the current attributes

See also

[setScale\(\)](#)

12.12.4.88 `double QwtDoubleRange::value () const [inherited]`

Returns the current value.

Reimplemented in [QwtCounter](#).

12.12.4.89 `void QwtDial::valueChange () [protected, virtual, inherited]`

[QwtDoubleRange](#) update hook.

Reimplemented from [QwtAbstractSlider](#).

12.12.4.90 `void QwtAbstractSlider::valueChanged (double value) [signal, inherited]`

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.12.4.91 `void QwtAbstractSlider::wheelEvent (QWheelEvent * e) [protected, virtual, inherited]`

Wheel Event handler

Parameters

<i>e</i>	Wheel event
----------	-------------

12.12.4.92 `bool QwtDial::wrapping () const [inherited]`

[wrapping\(\)](#) holds whether it is possible to step the value from the highest value to the lowest value and vice versa.

See also

[setWrapping\(\)](#), [QwtDoubleRange::setPeriodic\(\)](#)

Note

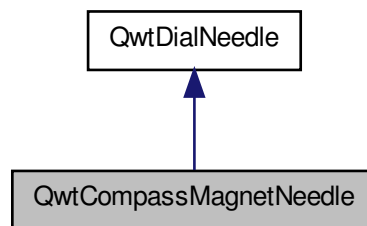
The meaning of wrapping is like the wrapping property of `QSpinBox`, but not like it is used in `QDial`.

12.13 QwtCompassMagnetNeedle Class Reference

A magnet needle for compass widgets.

```
#include <qwt_dial_needle.h>
```

Inheritance diagram for QwtCompassMagnetNeedle:



Public Types

- enum [Style](#) {
 TriangleStyle,
 ThinStyle }

Public Member Functions

- virtual void [draw](#) (QPainter *, const QPoint &, int length, double direction, QPalette::ColorGroup=QPalette::Active) const
- const QPalette & [palette](#) () const
- [QwtCompassMagnetNeedle](#) ([Style](#)=TriangleStyle, const QColor &light=Qt::white, const QColor &dark=Qt::red)
- virtual void [setPalette](#) (const QPalette &)

Static Public Member Functions

- static void [drawThinNeedle](#) (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)
- static void [drawTriangleNeedle](#) (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)

Static Protected Member Functions

- static void [drawKnob](#) (QPainter *, const QPoint &pos, int width, const QBrush &, bool sunken)
- static void [drawPointer](#) (QPainter *painter, const QBrush &brush, int colorOffset, const QPoint ¢er, int length, int width, double direction)

12.13.1 Detailed Description

A magnet needle for compass widgets. A magnet needle points to two opposite directions indicating north and south.

The following colors are used:

- QColorGroup::Light
Used for pointing south
- QColorGroup::Dark
Used for pointing north
- QColorGroup::Base
Knob (ThinStyle only)

See also

[QwtDial](#), [QwtCompass](#)

12.13.2 Member Enumeration Documentation

12.13.2.1 enum QwtCompassMagnetNeedle::Style

Style of the needle.

12.13.3 Constructor & Destructor Documentation

12.13.3.1 QwtCompassMagnetNeedle::QwtCompassMagnetNeedle (Style *style* = *TriangleStyle*, const QColor & *light* = Qt::white, const QColor & *dark* = Qt::red)

Constructor.

12.13.4 Member Function Documentation

12.13.4.1 `void QwtCompassMagnetNeedle::draw (QPainter * painter, const QPoint & center, int length, double direction, QPalette::ColorGroup colorGroup = QPalette::Active) const [virtual]`

Draw the needle

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial, start position for the needle
<i>length</i>	Length of the needle
<i>direction</i>	Direction of the needle, in degrees counter clockwise
<i>colorGroup</i>	Color group, used for painting

Implements [QwtDialNeedle](#).

12.13.4.2 `void QwtDialNeedle::drawKnob (QPainter * painter, const QPoint & pos, int width, const QBrush & brush, bool sunken) [static, protected, inherited]`

Draw the knob.

12.13.4.3 `void QwtCompassMagnetNeedle::drawPointer (QPainter * painter, const QBrush & brush, int colorOffset, const QPoint & center, int length, int width, double direction) [static, protected]`

Draw a compass needle

Parameters

<i>painter</i>	Painter
<i>brush</i>	Brush
<i>colorOffset</i>	Color offset
<i>center</i>	Center, where the needle starts
<i>length</i>	Length of the needle
<i>width</i>	Width of the needle
<i>direction</i>	Direction

12.13.4.4 `void QwtCompassMagnetNeedle::drawThinNeedle (QPainter * painter, const QPalette & palette, QPalette::ColorGroup colorGroup, const QPoint & center, int length, double direction) [static]`

Draw a compass needle

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>colorGroup</i>	Color group
<i>center</i>	Center, where the needle starts
<i>length</i>	Length of the needle
<i>direction</i>	Direction

12.13.4.5 `void QwtCompassMagnetNeedle::drawTriangleNeedle (QPainter *
painter, const QPalette & palette, QPalette::ColorGroup colorGroup,
const QPoint & center, int length, double direction) [static]`

Draw a compass needle

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>colorGroup</i>	Color group
<i>center</i>	Center, where the needle starts
<i>length</i>	Length of the needle
<i>direction</i>	Direction

12.13.4.6 `const QPalette & QwtDialNeedle::palette () const [inherited]`

Returns

the palette of the needle.

12.13.4.7 `void QwtDialNeedle::setPalette (const QPalette & palette)
[virtual, inherited]`

Sets the palette for the needle.

Parameters

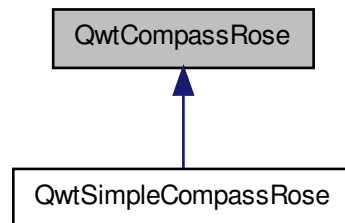
<i>palette</i>	New Palette
----------------	-------------

12.14 QwtCompassRose Class Reference

Abstract base class for a compass rose.

```
#include <qwt_compass_rose.h>
```

Inheritance diagram for QwtCompassRose:



Public Member Functions

- virtual void [draw](#) (QPainter *painter, const QPoint ¢er, int radius, double north, QPalette::ColorGroup colorGroup=QPalette::Active) const =0
- const QPalette & [palette](#) () const
- virtual void [setPalette](#) (const QPalette &p)

12.14.1 Detailed Description

Abstract base class for a compass rose.

12.14.2 Member Function Documentation

12.14.2.1 virtual void QwtCompassRose::draw (QPainter * *painter*, const QPoint & *center*, int *radius*, double *north*, QPalette::ColorGroup *colorGroup* = *QPalette::Active*) const [pure virtual]

Draw the rose

Parameters

<i>painter</i>	Painter
<i>center</i>	Center point
<i>radius</i>	Radius of the rose
<i>north</i>	Position
<i>colorGroup</i>	Color group

Implemented in [QwtSimpleCompassRose](#).

12.14.2.2 `const QPalette& QwtCompassRose::palette () const` `[inline]`

Returns

Current palette

12.14.2.3 `virtual void QwtCompassRose::setPalette (const QPalette & p)` `[inline, virtual]`

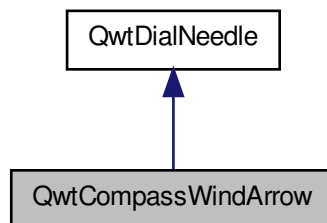
Assign a palette.

12.15 QwtCompassWindArrow Class Reference

An indicator for the wind direction.

```
#include <qwt_dial_needle.h>
```

Inheritance diagram for QwtCompassWindArrow:



Public Types

- enum `Style` {
 `Style1`,
 `Style2` }

Public Member Functions

- virtual void `draw` (QPainter *, const QPoint &, int length, double direction, QPalette::ColorGroup=QPalette::Active) const

- const QPalette & [palette](#) () const
- [QwtCompassWindArrow](#) ([Style](#), const QColor &light=Qt::white, const QColor &dark=Qt::gray)
- virtual void [setPalette](#) (const QPalette &)

Static Public Member Functions

- static void [drawStyle1Needle](#) (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)
- static void [drawStyle2Needle](#) (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)

Static Protected Member Functions

- static void [drawKnob](#) (QPainter *, const QPoint &pos, int width, const QBrush &, bool sunken)

12.15.1 Detailed Description

An indicator for the wind direction. [QwtCompassWindArrow](#) shows the direction where the wind comes from.

- QColorGroup::Light
Used for Style1, or the light half of Style2
- QColorGroup::Dark
Used for the dark half of Style2

See also

[QwtDial](#), [QwtCompass](#)

12.15.2 Member Enumeration Documentation

12.15.2.1 enum QwtCompassWindArrow::Style

Style of the arrow.

12.15.3 Constructor & Destructor Documentation

12.15.3.1 QwtCompassWindArrow::QwtCompassWindArrow (*Style style*, const QColor & *light* = *qt::white*, const QColor & *dark* = *qt::gray*)

Constructor

Parameters

<i>style</i>	Arrow style
<i>light</i>	Light color
<i>dark</i>	Dark color

12.15.4 Member Function Documentation

12.15.4.1 `void QwtCompassWindArrow::draw (QPainter * painter, const QPoint & center, int length, double direction, QPalette::ColorGroup colorGroup = QPalette::Active) const [virtual]`

Draw the needle

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial, start position for the needle
<i>length</i>	Length of the needle
<i>direction</i>	Direction of the needle, in degrees counter clockwise
<i>colorGroup</i>	Color group, used for painting

Implements [QwtDialNeedle](#).

12.15.4.2 `void QwtDialNeedle::drawKnob (QPainter * painter, const QPoint & pos, int width, const QBrush & brush, bool sunken) [static, protected, inherited]`

Draw the knob.

12.15.4.3 `void QwtCompassWindArrow::drawStyle1Needle (QPainter * painter, const QPalette & palette, QPalette::ColorGroup colorGroup, const QPoint & center, int length, double direction) [static]`

Draw a compass needle

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>colorGroup</i>	colorGroup
<i>center</i>	Center of the dial, start position for the needle
<i>length</i>	Length of the needle
<i>direction</i>	Direction of the needle, in degrees counter clockwise

12.15.4.4 void QwtCompassWindArrow::drawStyle2Needle (QPainter *
painter, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*,
const QPoint & *center*, int *length*, double *direction*) [static]

Draw a compass needle

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>colorGroup</i>	colorGroup
<i>center</i>	Center of the dial, start position for the needle
<i>length</i>	Length of the needle
<i>direction</i>	Direction of the needle, in degrees counter clockwise

12.15.4.5 const QPalette & QwtDialNeedle::palette () const [inherited]

Returns

the palette of the needle.

12.15.4.6 void QwtDialNeedle::setPalette (const QPalette & *palette*)
[virtual, inherited]

Sets the palette for the needle.

Parameters

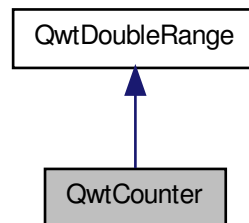
<i>palette</i>	New Palette
----------------	-------------

12.16 QwtCounter Class Reference

The Counter Widget.

```
#include <qwt_counter.h>
```


Inheritance diagram for QwtCounter:



Public Types

- enum [Button](#) {
 Button1,
 Button2,
 Button3,
 ButtonCnt }

Signals

- void [buttonReleased](#) (double value)
- void [valueChanged](#) (double value)

Public Member Functions

- bool [editable](#) () const
- virtual void [fitValue](#) (double)
- virtual void [incPages](#) (int)
- int [incSteps](#) ([QwtCounter::Button](#) btn) const
- virtual void [incValue](#) (int)
- bool [isValid](#) () const
- double [maxVal](#) () const
- double [maxValue](#) () const
- double [minVal](#) () const
- double [minValue](#) () const
- int [numButtons](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const

- virtual void [polish](#) ()
- [QwtCounter](#) (QWidget *parent=NULL)
- void [setEditable](#) (bool)
- void [setIncSteps](#) ([QwtCounter::Button](#) btn, int nSteps)
- void [setMaxValue](#) (double m)
- void [setMinValue](#) (double m)
- void [setNumButtons](#) (int n)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setStep](#) (double s)
- void [setStepButton1](#) (int nSteps)
- void [setStepButton2](#) (int nSteps)
- void [setStepButton3](#) (int nSteps)
- void [setValid](#) (bool)
- virtual void [setValue](#) (double)
- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- int [stepButton1](#) () const
- int [stepButton2](#) () const
- int [stepButton3](#) () const
- virtual double [value](#) () const
- virtual [~QwtCounter](#) ()

Protected Member Functions

- virtual bool [event](#) (QEvent *)
- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [keyPressEvent](#) (QKeyEvent *)
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- virtual void [stepChange](#) ()
- virtual void [wheelEvent](#) (QWheelEvent *)

12.16.1 Detailed Description

The Counter Widget. A Counter consists of a label displaying a number and one or more (up to three) push buttons on each side of the label which can be used to increment or decrement the counter's value.

A Counter has a range from a minimum value to a maximum value and a step size. The range can be specified using [QwtDbiRange::setRange\(\)](#). The counter's value is an integer multiple of the step size. The number of steps by which a button increments or decrements the value can be specified using [QwtCounter::setIncSteps\(\)](#). The number of buttons can be changed with [QwtCounter::setNumButtons\(\)](#).

Holding the space bar down with focus on a button is the fastest method to step through the counter values. When the counter underflows/overflows, the focus is set to the

smallest up/down button and counting is disabled. Counting is re-enabled on a button release event (mouse or space bar).

Example:

```
#include "../include/qwt_counter.h>

QwtCounter *cnt;

cnt = new QwtCounter(parent, name);

cnt->setRange(0.0, 100.0, 1.0);           // From 0.0 to 100, step 1.0
cnt->setNumButtons(2);                   // Two buttons each side
cnt->setIncSteps(QwtCounter::Button1, 1); // Button 1 increments 1 step
cnt->setIncSteps(QwtCounter::Button2, 20); // Button 2 increments 20 steps

connect(cnt, SIGNAL(valueChanged(double)), my_class, SLOT(newValue(double)));
```

12.16.2 Member Enumeration Documentation

12.16.2.1 enum QwtCounter::Button

Button index

12.16.3 Constructor & Destructor Documentation

12.16.3.1 QwtCounter::QwtCounter (QWidget * *parent* = *NULL*) [explicit]

The default number of buttons is set to 2. The default increments are:

- Button 1: 1 step
- Button 2: 10 steps
- Button 3: 100 steps

Parameters

<i>parent</i>

12.16.3.2 QwtCounter::~QwtCounter () [virtual]

Destructor.

12.16.4 Member Function Documentation

12.16.4.1 void QwtCounter::buttonReleased (double *value*) [signal]

This signal is emitted when a button has been released

Parameters

<i>value</i>	The new value
--------------	---------------

12.16.4.2 bool QwtCounter::editable () const

returns whether the line edit is editable. (default is yes)

12.16.4.3 bool QwtCounter::event (QEvent * *e*) [protected, virtual]

Handle PolishRequest events

12.16.4.4 double QwtDoubleRange::exactPrevValue () const [protected, inherited]

Returns the exact previous value.

12.16.4.5 double QwtDoubleRange::exactValue () const [protected, inherited]

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.16.4.6 void QwtDoubleRange::fitValue (double *x*) [virtual, inherited]

Adjust the value to the closest point in the step raster.

Parameters

<i>x</i>	value
----------	-------

Warning

The value is clipped when it lies outside the range. When the range is [QwtDoubleRange::periodic](#), it will be mapped to a point in the interval such that

$$\text{new value} := x + n * (\text{max. value} - \text{min. value})$$

with an integer number n .

Reimplemented in [QwtAbstractSlider](#).

12.16.4.7 void QwtDoubleRange::incPages (int *nPages*) [virtual, inherited]

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.16.4.8 int QwtCounter::incSteps (QwtCounter::Button *btn*) const

Returns

the number of steps by which a specified button increments the value or 0 if the button is invalid.

Parameters

<i>btn</i>	One of QwtCounter::Button1, QwtCounter::Button2, QwtCounter::Button3
------------	--

12.16.4.9 void QwtDoubleRange::incValue (int *nSteps*) [virtual, inherited]

Increment the value by a specified number of steps.

Parameters

<i>nSteps</i>	Number of steps to increment
---------------	------------------------------

Warning

As a result of this operation, the new value will always be adjusted to the step raster.

Reimplemented in [QwtAbstractSlider](#).

12.16.4.10 bool QwtDoubleRange::isValid () const [inherited]

Indicates if the value is valid.

Reimplemented in [QwtAbstractSlider](#).

**12.16.4.11 void QwtCounter::keyPressEvent (QKeyEvent * e)
[protected, virtual]**

Handle key events

- Ctrl + Qt::Key_Home Step to [minValue\(\)](#)
- Ctrl + Qt::Key_End Step to [maxValue\(\)](#)
- Qt::Key_Up Increment by [incSteps\(QwtCounter::Button1\)](#)
- Qt::Key_Down Decrement by [incSteps\(QwtCounter::Button1\)](#)
- Qt::Key_PageUp Increment by [incSteps\(QwtCounter::Button2\)](#)
- Qt::Key_PageDown Decrement by [incSteps\(QwtCounter::Button2\)](#)
- Shift + Qt::Key_PageUp Increment by [incSteps\(QwtCounter::Button3\)](#)
- Shift + Qt::Key_PageDown Decrement by [incSteps\(QwtCounter::Button3\)](#)

12.16.4.12 double QwtCounter::maxVal () const

returns the maximum value of the range

12.16.4.13 double QwtDoubleRange::maxValue () const [inherited]

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.16.4.14 double QwtCounter::minVal () const

returns the minimum value of the range

12.16.4.15 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.16.4.16 int QwtCounter::numButtons () const

Returns

The number of buttons on each side of the widget.

12.16.4.17 int QwtDoubleRange::pageSize () const [inherited]

Returns the page size in steps.

12.16.4.18 bool QwtDoubleRange::periodic () const [inherited]

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.16.4.19 void QwtCounter::polish () [virtual]

Sets the minimum width for the buttons

12.16.4.20 double QwtDoubleRange::prevValue () const [protected, inherited]

Returns the previous value.

12.16.4.21 void QwtCounter::rangeChange () [protected, virtual]

Notify change of range.

This function updates the enabled property of all buttons contained in [QwtCounter](#).

Reimplemented from [QwtDoubleRange](#).

12.16.4.22 void QwtCounter::setEditable (bool *editable*)

Allow/disallow the user to manually edit the value.

Parameters

<i>editable</i>	true enables editing
-----------------	----------------------

See also

[editable\(\)](#)

12.16.4.23 void QwtCounter::setIncSteps (QwtCounter::Button *btn*, int *nSteps*)

Specify the number of steps by which the value is incremented or decremented when a specified button is pushed.

Parameters

<i>btn</i>	One of QwtCounter::Button1, QwtCounter::Button2, QwtCounter::Button3
<i>nSteps</i>	Number of steps

12.16.4.24 void QwtCounter::setMaxValue (double *value*)

Set the maximum value of the range

Parameters

<i>value</i>	Maximum value
--------------	---------------

See also

[setMinValue\(\)](#), [maxVal\(\)](#)

12.16.4.25 void QwtCounter::setMinValue (double *value*)

Set the minimum value of the range

Parameters

<i>value</i>	Minimum value
--------------	---------------

See also

[setMaxValue\(\)](#), [minVal\(\)](#)

12.16.4.26 void QwtCounter::setNumButtons (int *n*)

Specify the number of buttons on each side of the label.

Parameters

<i>n</i>	Number of buttons
----------	-------------------

12.16.4.27 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.16.4.28 void QwtDoubleRange::setRange (double *vmin*, double *vmax*, double *vstep* = 0.0, int *pageSize* = 1) [inherited]

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will *not* be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.16.4.29 void QwtCounter::setStep (double *stepSize*)

Set the step size

Parameters

<i>stepSize</i>	Step size
-----------------	-----------

See also

[QwtDoubleRange::setStep\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.16.4.30 void QwtCounter::setStepButton1 (int *nSteps*)

Set the number of increment steps for button 1

Parameters

<i>nSteps</i>	Number of steps
---------------	-----------------

12.16.4.31 void QwtCounter::setStepButton2 (int *nSteps*)

Set the number of increment steps for button 2

Parameters

<i>nSteps</i>	Number of steps
---------------	-----------------

12.16.4.32 void QwtCounter::setStepButton3 (int *nSteps*)

Set the number of increment steps for button 3

Parameters

<i>nSteps</i>	Number of steps
---------------	-----------------

12.16.4.33 void QwtDoubleRange::setValid (bool *isValid*) [inherited]

Set the value to be valid/invalid.

Reimplemented in [QwtAbstractSlider](#).

12.16.4.34 void QwtCounter::setValue (double *v*) [virtual]

Set a new value.

Parameters

<i>v</i>	new value Calls QwtDoubleRange::setValue and does all visual updates.
----------	---

See also

[QwtDoubleRange::setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.16.4.35 QSize QwtCounter::sizeHint () const [virtual]

A size hint.

12.16.4.36 double QwtCounter::step () const

returns the step size

Reimplemented from [QwtDoubleRange](#).

12.16.4.37 int QwtCounter::stepButton1 () const

returns the number of increment steps for button 1

12.16.4.38 int QwtCounter::stepButton2 () const

returns the number of increment steps for button 2

12.16.4.39 int QwtCounter::stepButton3 () const

returns the number of increment steps for button 3

12.16.4.40 void QwtDoubleRange::stepChange () [protected, virtual, inherited]

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.16.4.41 double QwtCounter::value () const [virtual]**Returns**

Current value

Reimplemented from [QwtDoubleRange](#).

12.16.4.42 void QwtCounter::valueChanged (double *value*) [signal]

This signal is emitted when the counter's value has changed

Parameters

<i>value</i>	The new value
--------------	---------------

12.16.4.43 void QwtCounter::wheelEvent (QWheelEvent * *e*) [protected, virtual]

Handle wheel events

Parameters

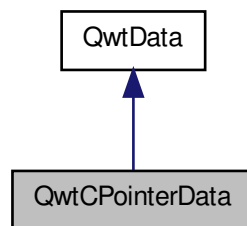
<i>e</i>	Wheel event
----------	-------------

12.17 QwtCPointerData Class Reference

Data class containing two pointers to memory blocks of doubles.

```
#include <qwt_data.h>
```

Inheritance diagram for QwtCPointerData:



Public Member Functions

- virtual QwtDoubleRect [boundingRect](#) () const
- virtual [QwtData](#) * [copy](#) () const
- [QwtCPointerData](#) & [operator=](#) (const [QwtCPointerData](#) &)
- [QwtCPointerData](#) (const double *x, const double *y, size_t size)
- virtual size_t [size](#) () const
- virtual double [x](#) (size_t i) const
- const double * [xData](#) () const
- virtual double [y](#) (size_t i) const
- const double * [yData](#) () const

12.17.1 Detailed Description

Data class containing two pointers to memory blocks of doubles.

12.17.2 Constructor & Destructor Documentation

12.17.2.1 QwtCPointerData::QwtCPointerData (const double * x, const double * y, size_t size)

Constructor

Parameters

<i>x</i>	Array of x values
<i>y</i>	Array of y values
<i>size</i>	Size of the x and y arrays

Warning

The programmer must assure that the memory blocks referenced by the pointers remain valid during the lifetime of the QwtPlotCPointer object.

See also

[QwtPlotCurve::setData\(\)](#), [QwtPlotCurve::setRawData\(\)](#)

12.17.3 Member Function Documentation**12.17.3.1 QwtDoubleRect QwtCPointerData::boundingRect () const [virtual]**

Returns the bounding rectangle of the data. If there is no bounding rect, like for empty data the rectangle is invalid: `QwtDoubleRect::isValid() == false`

Reimplemented from [QwtData](#).

12.17.3.2 QwtData * QwtCPointerData::copy () const [virtual]**Returns**

Pointer to a copy (virtual copy constructor)

Implements [QwtData](#).

12.17.3.3 QwtCPointerData & QwtCPointerData::operator= (const QwtCPointerData & data)

Assignment.

12.17.3.4 size_t QwtCPointerData::size () const [virtual]**Returns**

Size of the data set

Implements [QwtData](#).

12.17.3.5 double QwtCPointerData::x (size_t *i*) const [virtual]

Return the x value of data point *i*

Parameters

<i>i</i> Index

Returns

x X value of data point *i*

Implements [QwtData](#).

12.17.3.6 const double * QwtCPointerData::xData () const**Returns**

Array of the x-values

12.17.3.7 double QwtCPointerData::y (size_t *i*) const [virtual]

Return the y value of data point *i*

Parameters

<i>i</i> Index

Returns

y Y value of data point *i*

Implements [QwtData](#).

12.17.3.8 const double * QwtCPointerData::yData () const**Returns**

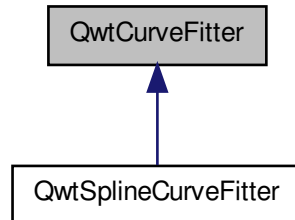
Array of the y-values

12.18 QwtCurveFitter Class Reference

Abstract base class for a curve fitter.

```
#include <qwt_curve_fitter.h>
```

Inheritance diagram for QwtCurveFitter:



Public Member Functions

- virtual QPolygonF [fitCurve](#) (const QPolygonF &polygon) const =0
- virtual [~QwtCurveFitter](#) ()

Protected Member Functions

- [QwtCurveFitter](#) ()

12.18.1 Detailed Description

Abstract base class for a curve fitter.

12.18.2 Constructor & Destructor Documentation

12.18.2.1 QwtCurveFitter::~QwtCurveFitter () [virtual]

Destructor.

12.18.2.2 QwtCurveFitter::QwtCurveFitter () [protected]

Constructor.

12.18.3 Member Function Documentation

12.18.3.1 virtual QPolygonF QwtCurveFitter::fitCurve (const QPolygonF & *polygon*) const [pure virtual]

Find a curve which has the best fit to a series of data points

Parameters

<i>polygon</i>	Series of data points
----------------	-----------------------

Returns

Curve points

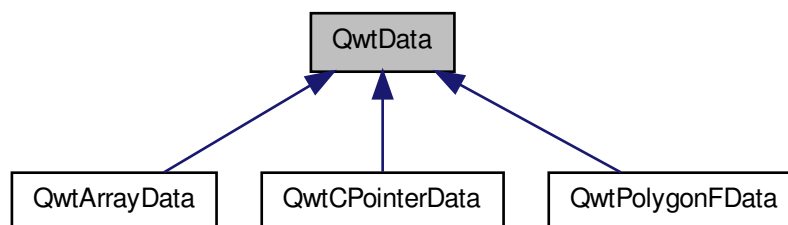
Implemented in [QwtSplineCurveFitter](#).

12.19 QwtData Class Reference

[QwtData](#) defines an interface to any type of curve data.

```
#include <qwt_data.h>
```

Inheritance diagram for QwtData:



Public Member Functions

- virtual QwtDoubleRect [boundingRect](#) () const
- virtual [QwtData](#) * [copy](#) () const =0
- [QwtData](#) ()
- virtual size_t [size](#) () const =0
- virtual double [x](#) (size_t i) const =0
- virtual double [y](#) (size_t i) const =0
- virtual [~QwtData](#) ()

Protected Member Functions

- [QwtData](#) & `operator=` (const [QwtData](#) &)

12.19.1 Detailed Description

[QwtData](#) defines an interface to any type of curve data. Classes, derived from [QwtData](#) may:

- store the data in almost any type of container
- calculate the data on the fly instead of storing it

12.19.2 Constructor & Destructor Documentation

12.19.2.1 `QwtData::QwtData ()`

Constructor.

12.19.2.2 `QwtData::~~QwtData () [virtual]`

Destructor.

12.19.3 Member Function Documentation

12.19.3.1 `QwtDoubleRect QwtData::boundingRect () const [virtual]`

Returns the bounding rectangle of the data. If there is no bounding rect, like for empty data the rectangle is invalid: `QwtDoubleRect::isValid() == false`

Warning

This is an slow implementation iterating over all points. It is intended to be over-loaded by derived classes. In case of auto scaling `boundingRect()` is called for every replot, so it might be worth to implement a cache, or use `x(0)`, `x(size() - 1)` for ordered data ...

Reimplemented in [QwtArrayData](#), and [QwtCPointerData](#).

12.19.3.2 `virtual QwtData* QwtData::copy () const [pure virtual]`

Returns

Pointer to a copy (virtual copy constructor)

Implemented in [QwtPolygonFData](#), [QwtArrayData](#), and [QwtCPointerData](#).

**12.19.3.3 QwtData& QwtData::operator= (const QwtData &)
[protected]**

Assignment operator (virtualized)

12.19.3.4 virtual size_t QwtData::size () const [pure virtual]
Returns

Size of the data set

Implemented in [QwtPolygonFData](#), [QwtArrayData](#), and [QwtCPointerData](#).

12.19.3.5 virtual double QwtData::x (size_t i) const [pure virtual]

Return the x value of data point i

Parameters

<i>i</i>	Index
----------	-------

Returns

x X value of data point i

Implemented in [QwtPolygonFData](#), [QwtArrayData](#), and [QwtCPointerData](#).

12.19.3.6 virtual double QwtData::y (size_t i) const [pure virtual]

Return the y value of data point i

Parameters

<i>i</i>	Index
----------	-------

Returns

y Y value of data point i

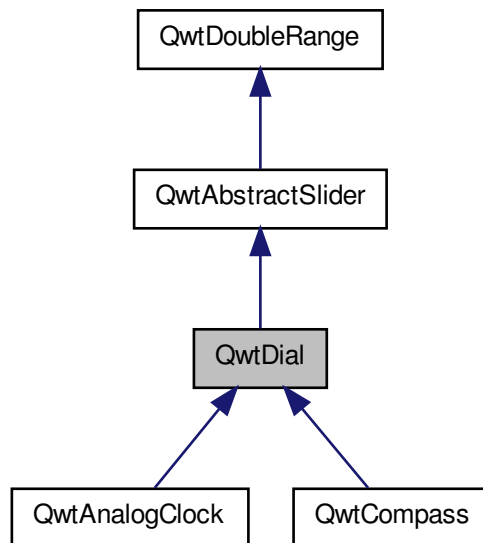
Implemented in [QwtPolygonFData](#), [QwtArrayData](#), and [QwtCPointerData](#).

12.20 QwtDial Class Reference

[QwtDial](#) class provides a rounded range control.

```
#include <qwt_dial.h>
```

Inheritance diagram for QwtDial:



Public Types

- enum [Direction](#) {
 Clockwise,
 CounterClockwise }
- enum [Mode](#) {
 RotateNeedle,
 RotateScale }
- enum [ScaleOptions](#) {
 ScaleBackbone = 1,
 ScaleTicks = 2,
 ScaleLabel = 4 }
- enum [ScrollMode](#) {
 ScrNone,

```

    ScrMouse,
    ScrTimer,
    ScrDirect,
    ScrPage }
• enum Shadow {
    Plain = QFrame::Plain,
    Raised = QFrame::Raised,
    Sunken = QFrame::Sunken }

```

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- virtual void [setReadOnly](#) (bool)
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()
- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- QRect [boundingRect](#) () const
- QRect [contentsRect](#) () const
- [Direction](#) [direction](#) () const
- [Shadow](#) [frameShadow](#) () const
- bool [hasVisibleBackground](#) () const
- virtual void [incPages](#) (int)
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- int [lineWidth](#) () const
- virtual double [mass](#) () const
- double [maxScaleArc](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const
- double [minScaleArc](#) () const
- double [minValue](#) () const
- [Mode](#) [mode](#) () const
- const [QwtDialNeedle](#) * [needle](#) () const
- [QwtDialNeedle](#) * [needle](#) ()
- Qt::Orientation [orientation](#) () const

- double [origin](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtDial](#) (QWidget *parent=NULL)
- virtual QRect [scaleContentsRect](#) () const
- [QwtDialScaleDraw](#) * [scaleDraw](#) ()
- const [QwtDialScaleDraw](#) * [scaleDraw](#) () const
- void [setDirection](#) (Direction)
- void [setFrameShadow](#) (Shadow)
- void [setLineWidth](#) (int)
- virtual void [setMass](#) (double val)
- void [setMode](#) (Mode)
- virtual void [setNeedle](#) (QwtDialNeedle *)
- virtual void [setOrientation](#) (Qt::Orientation o)
- virtual void [setOrigin](#) (double)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- virtual void [setScale](#) (int maxMajIntv, int maxMinIntv, double step=0.0)
- void [setScaleArc](#) (double min, double max)
- virtual void [setScaleDraw](#) (QwtDialScaleDraw *)
- void [setScaleOptions](#) (int)
- void [setScaleTicks](#) (int minLen, int medLen, int majLen, int penWidth=1)
- void [setStep](#) (double)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)
- virtual void [setWrapping](#) (bool)
- void [showBackground](#) (bool)
- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- void [stopMoving](#) ()
- double [value](#) () const
- bool [wrapping](#) () const
- virtual ~[QwtDial](#) ()

Protected Member Functions

- virtual void [drawContents](#) (QPainter *) const
- virtual void [drawFocusIndicator](#) (QPainter *) const
- virtual void [drawFrame](#) (QPainter *p)
- virtual void [drawNeedle](#) (QPainter *, const QPoint &, int radius, double direction, QPalette::ColorGroup) const
- virtual void [drawScale](#) (QPainter *, const QPoint ¢er, int radius, double origin, double arcMin, double arcMax) const
- virtual void [drawScaleContents](#) (QPainter *painter, const QPoint ¢er, int radius) const

- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [getScrollMode](#) (const QPoint &, int &scrollMode, int &direction)
- virtual double [getValue](#) (const QPoint &)
- virtual void [keyPressEvent](#) (QKeyEvent *)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)
- double [mouseOffset](#) () const
- virtual void [mousePressEvent](#) (QMouseEvent *e)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *e)
- virtual void [paintEvent](#) (QPaintEvent *)
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- virtual void [resizeEvent](#) (QResizeEvent *)
- virtual [QwtText scaleLabel](#) (double) const
- int [scrollMode](#) () const
- void [setMouseOffset](#) (double)
- virtual void [setPosition](#) (const QPoint &)
- virtual void [stepChange](#) ()
- virtual void [timerEvent](#) (QTimerEvent *e)
- virtual void [updateMask](#) ()
- void [updateScale](#) ()
- virtual void [valueChange](#) ()
- virtual void [wheelEvent](#) (QWheelEvent *e)

Friends

- class [QwtDialScaleDraw](#)

12.20.1 Detailed Description

[QwtDial](#) class provides a rounded range control. [QwtDial](#) is intended as base class for dial widgets like speedometers, compass widgets, clocks ...

A dial contains a scale and a needle indicating the current value of the dial. Depending on Mode one of them is fixed and the other is rotating. If not [isReadOnly\(\)](#) the dial can be rotated by dragging the mouse or using keyboard inputs (see [keyPressEvent\(\)](#)). A dial might be wrapping, what means a rotation below/above one limit continues on the other limit (f.e compass). The scale might cover any arc of the dial, its values are related to the [origin\(\)](#) of the dial.

Qwt is missing a set of good looking needles ([QwtDialNeedle](#)). Contributions are very welcome.

See also

[QwtCompass](#), [QwtAnalogClock](#), [QwtDialNeedle](#)

Note

The examples/dials example shows different types of dials.

12.20.2 Member Enumeration Documentation**12.20.2.1 enum QwtDial::Direction**

Direction of the dial

12.20.2.2 enum QwtDial::Mode

In case of RotateNeedle the needle is rotating, in case of RotateScale, the needle points to [origin\(\)](#) and the scale is rotating.

12.20.2.3 enum QwtDial::ScaleOptions

see [QwtDial::setScaleOptions](#)

12.20.2.4 enum QwtAbstractSlider::ScrollMode [inherited]

Scroll mode

See also

[getScrollMode\(\)](#)

12.20.2.5 enum QwtDial::Shadow

Frame shadow.

Unfortunately it is not possible to use QFrame::Shadow as a property of a widget that is not derived from QFrame. The following enum is made for the designer only. It is safe to use QFrame::Shadow instead.

12.20.3 Constructor & Destructor Documentation**12.20.3.1 QwtDial::QwtDial (QWidget * *parent* = *NULL*) [explicit]**

Constructor.

Parameters

<i>parent</i>	Parent widget
---------------	---------------

Create a dial widget with no scale and no needle. The default origin is 90.0 with no valid value. It accepts mouse and keyboard inputs and has no step size. The default

mode is QwtDial::RotateNeedle.

12.20.3.2 QwtDial::~~QwtDial () [virtual]

Destructor.

12.20.4 Member Function Documentation

12.20.4.1 QRect QwtDial::boundingRect () const

Returns

bounding rect of the dial including the frame

See also

[setLineWidth\(\)](#), [scaleContentsRect\(\)](#), [contentsRect\(\)](#)

12.20.4.2 QRect QwtDial::contentsRect () const

Returns

bounding rect of the circle inside the frame

See also

[setLineWidth\(\)](#), [scaleContentsRect\(\)](#), [boundingRect\(\)](#)

12.20.4.3 QwtDial::Direction QwtDial::direction () const

Returns

Direction of the dial

The default direction of a dial is QwtDial::Clockwise

See also

[setDirection\(\)](#)

12.20.4.4 void QwtDial::drawContents (QPainter * *painter*) const
[protected, virtual]

Draw the contents inside the frame.

QColorGroup::Background is the background color outside of the frame. QColorGroup::Base is the background color inside the frame. QColorGroup::Foreground is the background color inside the scale.

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[boundingRect\(\)](#), [contentsRect\(\)](#), [scaleContentsRect\(\)](#), [QWidget::setPalette\(\)](#)

12.20.4.5 void QwtDial::drawFocusIndicator (QPainter * *painter*) const
[protected, virtual]

Draw a dotted round circle, if `!isReadOnly()`

Parameters

<i>painter</i>	Painter
----------------	---------

12.20.4.6 void QwtDial::drawFrame (QPainter * *painter*) [protected, virtual]

Draw the frame around the dial

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[lineWidth\(\)](#), [frameShadow\(\)](#)

12.20.4.7 void QwtDial::drawNeedle (QPainter * *painter*, const QPoint & *center*, int *radius*, double *direction*, QPalette::ColorGroup *cg*) const
[protected, virtual]

Draw the needle

Parameters

--	--

<i>painter</i>	Painter
<i>center</i>	Center of the dial
<i>radius</i>	Length for the needle
<i>direction</i>	Direction of the needle in degrees, counter clockwise
<i>cg</i>	ColorGroup

Reimplemented in [QwtAnalogClock](#).

12.20.4.8 `void QwtDial::drawScale (QPainter * painter, const QPoint & center, int radius, double origin, double minArc, double maxArc) const [protected, virtual]`

Draw the scale

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial
<i>radius</i>	Radius of the scale
<i>origin</i>	Origin of the scale
<i>minArc</i>	Minimum of the arc
<i>maxArc</i>	Maximum of the arc

See also

[QwtAbstractScaleDraw::setAngleRange\(\)](#)

12.20.4.9 `void QwtDial::drawScaleContents (QPainter * painter, const QPoint & center, int radius) const [protected, virtual]`

Draw the contents inside the scale

Paints nothing.

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the contents circle
<i>radius</i>	Radius of the contents circle

Reimplemented in [QwtCompass](#).

12.20.4.10 `double QwtDoubleRange::exactPrevValue () const [protected, inherited]`

Returns the exact previous value.

12.20.4.11 `double QwtDoubleRange::exactValue () const` [**protected**, **inherited**]

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.20.4.12 `void QwtAbstractSlider::fitValue (double value)` [**virtual**, **slot**, **inherited**]

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.20.4.13 `QwtDial::Shadow QwtDial::frameShadow () const`

Returns

Frame shadow /sa [setFrameShadow\(\)](#), [lineWidth\(\)](#), [QFrame::frameShadow](#)

12.20.4.14 `void QwtDial::getScrollMode (const QPoint & pos, int & scrollMode, int & direction)` [**protected**, **virtual**]

See [QwtAbstractSlider::getScrollMode\(\)](#)

Parameters

<i>pos</i>	point where the mouse was pressed
------------	-----------------------------------

Return values

<i>scrollMode</i>	The scrolling mode
<i>direction</i>	direction: 1, 0, or -1.

See also

[QwtAbstractSlider::getScrollMode\(\)](#)

Implements [QwtAbstractSlider](#).

12.20.4.15 **double QwtDial::getValue (const QPoint & *pos*) [protected, virtual]**

Find the value for a given position

Parameters

<i>pos</i>	Position
------------	----------

Returns

Value

Implements [QwtAbstractSlider](#).

12.20.4.16 **bool QwtDial::hasVisibleBackground () const**

true when the area outside of the frame is visible

See also

[showBackground\(\)](#), [setMask\(\)](#)

12.20.4.17 **void QwtDoubleRange::incPages (int *nPages*) [virtual, inherited]**

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.20.4.18 `void QwtAbstractSlider::incValue (int steps) [virtual, slot, inherited]`

Increment the value by a specified number of steps.

Parameters

<i>steps</i>	number of steps
--------------	-----------------

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.20.4.19 `bool QwtAbstractSlider::isReadOnly () const [inherited]`

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.20.4.20 `bool QwtAbstractSlider::isValid () const [inline, inherited]`

See also

[QwtDbfRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.20.4.21 `void QwtDial::keyPressEvent (QKeyEvent * event) [protected, virtual]`

Handles key events

- Key_Down, KeyLeft
Decrement by 1
- Key_Prior
Decrement by [pageSize\(\)](#)

- Key_Home
Set the value to [minValue\(\)](#)
- Key_Up, KeyRight
Increment by 1
- Key_Next
Increment by [pageSize\(\)](#)
- Key_End
Set the value to [maxValue\(\)](#)

Parameters

<i>event</i>	Key event
--------------	-----------

See also

[isReadOnly\(\)](#)

Reimplemented from [QwtAbstractSlider](#).

Reimplemented in [QwtCompass](#).

12.20.4.22 int QwtDial::lineWidth () const**Returns**

Line width of the frame

See also

[setLineWidth\(\)](#), [frameShadow\(\)](#), [lineWidth\(\)](#)

12.20.4.23 double QwtAbstractSlider::mass () const [virtual, inherited]**Returns**

mass

See also

[setMass\(\)](#)

Reimplemented in [QwtWheel](#).

12.20.4.24 double QwtDial::maxScaleArc () const**Returns**

Upper limit of the scale arc

12.20.4.25 double QwtDoubleRange::maxValue () const [inherited]

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.20.4.26 QSize QwtDial::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

The return value of [QwtDial::minimumSizeHint\(\)](#) depends on the font and the scale.

12.20.4.27 double QwtDial::minScaleArc () const**Returns**

Lower limit of the scale arc

12.20.4.28 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.20.4.29 QwtDial::Mode QwtDial::mode () const**Returns**

mode of the dial.

The value of the dial is indicated by the difference between the origin and the direction of the needle. In case of `QwtDial::RotateNeedle` the scale arc is fixed to the [origin\(\)](#) and the needle is rotating, in case of `QwtDial::RotateScale`, the needle points to [origin\(\)](#) and the scale is rotating.

The default mode is `QwtDial::RotateNeedle`.

See also

[setMode\(\)](#), [origin\(\)](#), [setScaleArc\(\)](#), [value\(\)](#)

**12.20.4.30 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

**12.20.4.31 void QwtAbstractSlider::mousePressEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

**12.20.4.32 void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.20.4.33 QwtDialNeedle * QwtDial::needle ()**Returns**

needle

See also

[setNeedle\(\)](#)

12.20.4.34 const QwtDialNeedle * QwtDial::needle () const**Returns**

needle

See also

[setNeedle\(\)](#)

**12.20.4.35 Qt::Orientation QwtAbstractSlider::orientation () const
[inherited]****Returns**

Orientation

See also

[setOrientation\(\)](#)

12.20.4.36 double QwtDial::origin () const

The origin is the angle where scale and needle is relative to.

Returns

Origin of the dial

See also

[setOrigin\(\)](#)

12.20.4.37 `int QwtDoubleRange::pageSize () const [inherited]`

Returns the page size in steps.

12.20.4.38 `void QwtDial::paintEvent (QPaintEvent * e) [protected, virtual]`

Paint the dial

Parameters

<i>e</i>	Paint event
----------	-------------

12.20.4.39 `bool QwtDoubleRange::periodic () const [inherited]`

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.20.4.40 `double QwtDoubleRange::prevValue () const [protected, inherited]`

Returns the previous value.

12.20.4.41 `void QwtDial::rangeChange () [protected, virtual]`

[QwtDoubleRange](#) update hook.

Reimplemented from [QwtDoubleRange](#).

12.20.4.42 `void QwtDial::resizeEvent (QResizeEvent * e) [protected, virtual]`

Resize the dial widget

Parameters

<i>e</i>	Resize event
----------	--------------

12.20.4.43 `QRect QwtDial::scaleContentsRect () const [virtual]`**Returns**

rect inside the scale

See also

[setLineWidth\(\)](#), [boundingRect\(\)](#), [contentsRect\(\)](#)

12.20.4.44 `QwtDialScaleDraw * QwtDial::scaleDraw ()`

Return the scale draw.

12.20.4.45 `const QwtDialScaleDraw * QwtDial::scaleDraw () const`

Return the scale draw.

12.20.4.46 `QwtText QwtDial::scaleLabel (double value) const [protected, virtual]`

Find the label for a value

Parameters

<i>value</i>	Value
--------------	-------

Returns

label

Reimplemented in [QwtAnalogClock](#), and [QwtCompass](#).

12.20.4.47 `void QwtDial::setDirection (Direction direction)`

Set the direction of the dial (clockwise/counterclockwise)

Direction *direction*

See also

[direction\(\)](#)

12.20.4.48 void QwtDial::setFrameShadow (Shadow *shadow*)

Sets the frame shadow value from the frame style.

Parameters

<i>shadow</i>	Frame shadow
---------------	--------------

See also

[setLineWidth\(\)](#), [QFrame::setFrameShadow\(\)](#)

12.20.4.49 void QwtDial::setLineWidth (int *lineWidth*)

Sets the line width

Parameters

<i>lineWidth</i>	Line width
------------------	------------

See also

[setFrameShadow\(\)](#)

12.20.4.50 void QwtAbstractSlider::setMass (double *val*) [virtual, inherited]

Set the slider's mass for flywheel effect.

If the slider's mass is greater then 0, it will continue to move after the mouse button has been released. Its speed decreases with time at a rate depending on the slider's mass. A large mass means that it will continue to move for a long time.

Derived widgets may overload this function to make it public.

Parameters

<i>val</i>	New mass in kg
------------	----------------

See also

[mass\(\)](#)

Reimplemented in [QwtWheel](#).

12.20.4.51 void QwtDial::setMode (Mode *mode*)

Change the mode of the meter.

Parameters

<i>mode</i>	New mode
-------------	----------

The value of the meter is indicated by the difference between north of the scale and the direction of the needle. In case of `QwtDial::RotateNeedle` north is pointing to the [origin\(\)](#) and the needle is rotating, in case of `QwtDial::RotateScale`, the needle points to [origin\(\)](#) and the scale is rotating.

The default mode is `QwtDial::RotateNeedle`.

See also

[mode\(\)](#), [setValue\(\)](#), [setOrigin\(\)](#)

12.20.4.52 void QwtDial::setNeedle (QwtDialNeedle * *needle*) [virtual]

Set a needle for the dial

Qwt is missing a set of good looking needles. Contributions are very welcome.

Parameters

<i>needle</i>	Needle
---------------	--------

Warning

The needle will be deleted, when a different needle is set or in [~QwtDial\(\)](#)

12.20.4.53 void QwtAbstractSlider::setOrientation (Qt::Orientation *o*) [virtual, inherited]

Set the orientation.

Parameters

<i>o</i>	Orientation. Allowed values are <code>Qt::Horizontal</code> and <code>Qt::Vertical</code> .
----------	---

Reimplemented in [QwtSlider](#), and [QwtWheel](#).

12.20.4.54 void QwtDial::setOrigin (double *origin*) [virtual]

Change the origin.

The origin is the angle where scale and needle is relative to.

Parameters

<i>origin</i>	New origin
---------------	------------

See also

[origin\(\)](#)

12.20.4.55 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.20.4.56 void QwtAbstractSlider::setPosition (const QPoint & *p*) [protected, virtual, inherited]

Move the slider to a specified point, adjust the value and emit signals if necessary.

12.20.4.57 void QwtDoubleRange::setRange (double *vmin*, double *vmax*, double *vstep* = 0.0, int *pageSize* = 1) [inherited]

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The

current value will **not** be adjusted to the new step raster.

- $v_{\max} < v_{\min}$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.20.4.58 `void QwtAbstractSlider::setReadOnly (bool readOnly)`
[virtual, slot, inherited]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also

[isReadOnly\(\)](#)

12.20.4.59 `void QwtDial::setScale (int maxMajIntv, int maxMinIntv, double step = 0.0)`
[virtual]

Change the intervals of the scale

See also

`QwtAbstractScaleDraw::setScale()`

12.20.4.60 `void QwtDial::setScaleArc (double minArc, double maxArc)`

Change the arc of the scale

Parameters

<i>minArc</i>	Lower limit
<i>maxArc</i>	Upper limit

12.20.4.61 `void QwtDial::setScaleDraw (QwtDialScaleDraw * scaleDraw)`
[virtual]

Set an individual scale draw

Parameters

<i>scaleDraw</i>	Scale draw
------------------	------------

Warning

The previous scale draw is deleted

12.20.4.62 void QwtDial::setScaleOptions (int *options*)

A wrapper method for accessing the scale draw.

- *options* == 0
No visible scale: `setScaleDraw(NULL)`
- *options* & `ScaleBackbone`
En/disable the backbone of the scale.
- *options* & `ScaleTicks`
En/disable the ticks of the scale.
- *options* & `ScaleLabel`
En/disable scale labels

See also

[QwtAbstractScaleDraw::enableComponent\(\)](#)

12.20.4.63 void QwtDial::setScaleTicks (int *minLen*, int *medLen*, int *majLen*, int *penWidth* = 1)

Assign length and width of the ticks

Parameters

<i>minLen</i>	Length of the minor ticks
<i>medLen</i>	Length of the medium ticks
<i>majLen</i>	Length of the major ticks
<i>penWidth</i>	Width of the pen for all ticks

See also

[QwtAbstractScaleDraw::setTickLength\(\)](#), [QwtDialScaleDraw::setPenWidth\(\)](#)

12.20.4.64 void QwtDoubleRange::setStep (double *vstep*) [inherited]

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

**12.20.4.65 void QwtAbstractSlider::setTracking (bool *enable*)
[inherited]**

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i>	true (enable) or false (disable) tracking.
---------------	--

12.20.4.66 void QwtAbstractSlider::setUpdateTime (int *t*) [inherited]

Specify the update interval for automatic scrolling.

Parameters

<i>t</i>	update interval in milliseconds
----------	---------------------------------

See also

[getScrollMode\(\)](#)

12.20.4.67 void QwtAbstractSlider::setValid (bool *valid*) [**inline**, **inherited**]

Parameters

<i>valid</i> true/false

See also

QwtDblRange::isValid()

Reimplemented from [QwtDoubleRange](#).

12.20.4.68 void QwtAbstractSlider::setValue (double *val*) [**virtual**, **slot**, **inherited**]

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i> new value

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.20.4.69 void QwtDial::setWrapping (bool *wrapping*) [**virtual**]

Sets whether it is possible to step the value from the highest value to the lowest value and vice versa to on.

Parameters

<i>wrapping</i> en/disables wrapping

See also

[wrapping\(\)](#), [QwtDoubleRange::periodic\(\)](#)

Note

The meaning of wrapping is like the wrapping property of QSpinBox, but not like it is used in QDial.

12.20.4.70 void QwtDial::showBackground (bool *show*)

Show/Hide the area outside of the frame

Parameters

<i>show</i>	Show if true, hide if false
-------------	-----------------------------

See also

[hasVisibleBackground\(\)](#), [setMask\(\)](#)

Warning

When [QwtDial](#) is a toplevel widget the window border might disappear too.

12.20.4.71 QSize QwtDial::sizeHint () const [virtual]**Returns**

Size hint

12.20.4.72 void QwtAbstractSlider::sliderMoved (double *value*) [signal, inherited]

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i>	new value
--------------	-----------

12.20.4.73 void QwtAbstractSlider::sliderPressed () [signal, inherited]

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.20.4.74 void QwtAbstractSlider::sliderReleased () [signal, inherited]

This signal is emitted when the user releases the movable part of the slider.

12.20.4.75 double QwtDoubleRange::step () const [inherited]

Returns

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.20.4.76 `void QwtDoubleRange::stepChange () [protected, virtual, inherited]`

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.20.4.77 `void QwtAbstractSlider::stopMoving () [inherited]`

Stop updating if automatic scrolling is active.

12.20.4.78 `void QwtAbstractSlider::timerEvent (QTimerEvent * e) [protected, virtual, inherited]`

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.20.4.79 `void QwtDial::updateMask () [protected, virtual]`

Update the mask of the dial.

In case of "hasVisibleBackground() == false", the background is transparent by a mask.

See also

[showBackground\(\)](#), [hasVisibleBackground\(\)](#)

12.20.4.80 `void QwtDial::updateScale () [protected]`

Update the scale with the current attributes

See also

[setScale\(\)](#)

12.20.4.81 double QwtDoubleRange::value () const [inherited]

Returns the current value.

Reimplemented in [QwtCounter](#).

12.20.4.82 void QwtDial::valueChange () [protected, virtual]

[QwtDoubleRange](#) update hook.

Reimplemented from [QwtAbstractSlider](#).

12.20.4.83 void QwtAbstractSlider::valueChanged (double *value*) [signal, inherited]

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.20.4.84 void QwtAbstractSlider::wheelEvent (QWheelEvent * *e*) [protected, virtual, inherited]

Wheel Event handler

Parameters

<i>e</i>	Wheel event
----------	-------------

12.20.4.85 bool QwtDial::wrapping () const

[wrapping\(\)](#) holds whether it is possible to step the value from the highest value to the lowest value and vice versa.

See also

[setWrapping\(\)](#), [QwtDoubleRange::setPeriodic\(\)](#)

Note

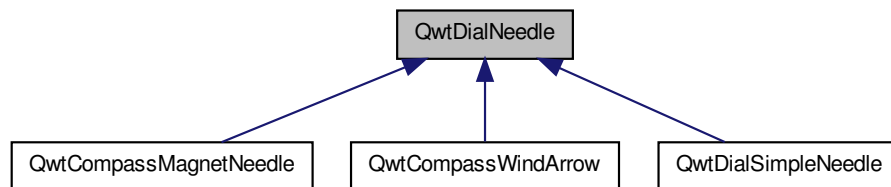
The meaning of wrapping is like the wrapping property of [QSpinBox](#), but not like it is used in [QDial](#).

12.21 QwtDialNeedle Class Reference

Base class for needles that can be used in a [QwtDial](#).

```
#include <qwt_dial_needle.h>
```

Inheritance diagram for QwtDialNeedle:

**Public Member Functions**

- virtual void [draw](#) (QPainter *painter, const QPoint ¢er, int length, double direction, QPalette::ColorGroup cg=QPalette::Active) const =0
- const QPalette & [palette](#) () const
- [QwtDialNeedle](#) ()
- virtual void [setPalette](#) (const QPalette &)
- virtual [~QwtDialNeedle](#) ()

Static Protected Member Functions

- static void [drawKnob](#) (QPainter *, const QPoint &pos, int width, const QBrush &, bool sunken)

12.21.1 Detailed Description

Base class for needles that can be used in a [QwtDial](#). [QwtDialNeedle](#) is a pointer that indicates a value by pointing to a specific direction.

Qwt is missing a set of good looking needles. Contributions are very welcome.

See also

[QwtDial](#), [QwtCompass](#)

12.21.2 Constructor & Destructor Documentation

12.21.2.1 QwtDialNeedle::QwtDialNeedle ()

Constructor.

12.21.2.2 QwtDialNeedle::~~QwtDialNeedle () [virtual]

Destructor.

12.21.3 Member Function Documentation

12.21.3.1 virtual void QwtDialNeedle::draw (QPainter * *painter*, const QPoint & *center*, int *length*, double *direction*, QPalette::ColorGroup *cg* = QPalette::Active) const [pure virtual]

Draw the needle

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial, start position for the needle
<i>length</i>	Length of the needle
<i>direction</i>	Direction of the needle, in degrees counter clockwise
<i>cg</i>	Color group, used for painting

Implemented in [QwtDialSimpleNeedle](#), [QwtCompassMagnetNeedle](#), and [QwtCompassWindArrow](#).

12.21.3.2 void QwtDialNeedle::drawKnob (QPainter * *painter*, const QPoint & *pos*, int *width*, const QBrush & *brush*, bool *sunken*) [static, protected]

Draw the knob.

12.21.3.3 const QPalette & QwtDialNeedle::palette () const

Returns

the palette of the needle.

12.21.3.4 `void QwtDialNeedle::setPalette (const QPalette & palette)`
[virtual]

Sets the palette for the needle.

Parameters

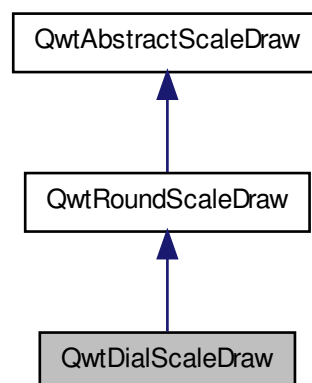
<i>palette</i>	New Palette
----------------	-------------

12.22 QwtDialScaleDraw Class Reference

A special scale draw made for [QwtDial](#).

```
#include <qwt_dial.h>
```

Inheritance diagram for QwtDialScaleDraw:

**Public Types**

- enum [ScaleComponent](#) {
 Backbone = 1,
 Ticks = 2,
 Labels = 4 }

Public Member Functions

- [QPoint](#) [center](#) () const
- virtual void [draw](#) (QPainter *, const QPalette &) const
- void [enableComponent](#) ([ScaleComponent](#), bool enable=true)
- virtual int [extent](#) (const QPen &, const QFont &) const
- bool [hasComponent](#) ([ScaleComponent](#)) const
- virtual [QwtText](#) [label](#) (double value) const
- int [majTickLength](#) () const
- const [QwtScaleMap](#) & [map](#) () const
- int [minimumExtent](#) () const
- void [moveCenter](#) (const QPoint &)
- void [moveCenter](#) (int x, int y)
- uint [penWidth](#) () const
- [QwtDialScaleDraw](#) ([QwtDial](#) *)
- int [radius](#) () const
- const [QwtScaleDiv](#) & [scaleDiv](#) () const
- [QwtScaleMap](#) & [scaleMap](#) ()
- void [setAngleRange](#) (double angle1, double angle2)
- void [setMinimumExtent](#) (int)
- void [setPenWidth](#) (uint)
- void [setRadius](#) (int radius)
- void [setScaleDiv](#) (const [QwtScaleDiv](#) &s)
- void [setSpacing](#) (int margin)
- void [setTickLength](#) ([QwtScaleDiv::TickType](#), int length)
- void [setTransformation](#) ([QwtScaleTransformation](#) *)
- int [spacing](#) () const
- int [tickLength](#) ([QwtScaleDiv::TickType](#)) const

Protected Member Functions

- virtual void [drawBackbone](#) (QPainter *p) const
- virtual void [drawLabel](#) (QPainter *p, double val) const
- virtual void [drawTick](#) (QPainter *p, double val, int len) const
- void [invalidateCache](#) ()
- const [QwtText](#) & [tickLabel](#) (const QFont &, double value) const

12.22.1 Detailed Description

A special scale draw made for [QwtDial](#).

See also

[QwtDial](#), [QwtCompass](#)

12.22.2 Member Enumeration Documentation

12.22.2.1 enum QwtAbstractScaleDraw::ScaleComponent [inherited]

Components of a scale

- Backbone
- Ticks
- Labels

See also

[enableComponent\(\)](#), [hasComponent](#)

12.22.3 Constructor & Destructor Documentation

12.22.3.1 QwtDialScaleDraw::QwtDialScaleDraw (QwtDial * *parent*)
[explicit]

Constructor

Parameters

<i>parent</i>	Parent dial widget
---------------	--------------------

12.22.4 Member Function Documentation

12.22.4.1 QPoint QwtRoundScaleDraw::center () const [inherited]

Get the center of the scale.

12.22.4.2 void QwtAbstractScaleDraw::draw (QPainter * *painter*, const
QPalette & *palette*) const [virtual, inherited]

Draw the scale.

Parameters

<i>painter</i>	The painter
<i>palette</i>	Palette, text color is used for the labels, foreground color for ticks and backbone

12.22.4.3 void QwtRoundScaleDraw::drawBackbone (QPainter * *painter*)
const [protected, virtual, inherited]

Draws the baseline of the scale

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[drawTick\(\)](#), [drawLabel\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.22.4.4 void QwtRoundScaleDraw::drawLabel (QPainter * *painter*, double
value) const [protected, virtual, inherited]

Draws the label for a major scale tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value

See also

[drawTick\(\)](#), [drawBackbone\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.22.4.5 void QwtRoundScaleDraw::drawTick (QPainter * *painter*, double
value, int *len*) const [protected, virtual, inherited]

Draw a tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value of the tick
<i>len</i>	Length of the tick

See also

[drawBackbone\(\)](#), [drawLabel\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.22.4.6 `void QwtAbstractScaleDraw::enableComponent (ScaleComponent
component, bool enable = true) [inherited]`

En/Disable a component of the scale

Parameters

<i>component</i>	Scale component
<i>enable</i>	On/Off

See also

[hasComponent\(\)](#)

12.22.4.7 `int QwtRoundScaleDraw::extent (const QPen & pen, const QFont &
font) const [virtual, inherited]`

Calculate the extent of the scale

The extent is the distance between the baseline to the outermost pixel of the scale draw. [radius\(\)](#) + [extent\(\)](#) is an upper limit for the radius of the bounding circle.

Parameters

<i>pen</i>	Pen that is used for painting backbone and ticks
<i>font</i>	Font used for painting the labels

See also

[setMinimumExtent\(\)](#), [minimumExtent\(\)](#)

Warning

The implemented algo is not too smart and calculates only an upper limit, that might be a few pixels too large

Implements [QwtAbstractScaleDraw](#).

12.22.4.8 `bool QwtAbstractScaleDraw::hasComponent (ScaleComponent
component) const [inherited]`

Check if a component is enabled

See also

[enableComponent\(\)](#)

12.22.4.9 void QwtAbstractScaleDraw::invalidateCache () [protected, inherited]

Invalidate the cache used by [QwtAbstractScaleDraw::tickLabel](#)

The cache is invalidated, when a new [QwtScaleDiv](#) is set. If the labels need to be changed. while the same [QwtScaleDiv](#) is set, [QwtAbstractScaleDraw::invalidateCache](#) needs to be called manually.

12.22.4.10 QwtText QwtDialScaleDraw::label (double *value*) const [virtual]

Call [QwtDial::scaleLabel](#) of the parent dial widget.

Parameters

<i>value</i>	Value to display
--------------	------------------

See also

[QwtDial::scaleLabel\(\)](#)

Reimplemented from [QwtAbstractScaleDraw](#).

12.22.4.11 int QwtAbstractScaleDraw::majTickLength () const [inherited]

The same as [QwtAbstractScaleDraw::tickLength\(QwtScaleDiv::MajorTick\)](#).

12.22.4.12 const QwtScaleMap & QwtAbstractScaleDraw::map () const [inherited]

Returns

Map how to translate between scale and pixel values

12.22.4.13 int QwtAbstractScaleDraw::minimumExtent () const [inherited]

Get the minimum extent

See also

[extent\(\)](#), [setMinimumExtent\(\)](#)

12.22.4.14 `void QwtRoundScaleDraw::moveCenter (int x, int y) [inline, inherited]`

Move the center of the scale draw, leaving the radius unchanged.

12.22.4.15 `void QwtRoundScaleDraw::moveCenter (const QPoint & center) [inherited]`

Move the center of the scale draw, leaving the radius unchanged

Parameters

<i>center</i>	New center
---------------	------------

See also

[setRadius\(\)](#)

12.22.4.16 `uint QwtDialScaleDraw::penWidth () const`

Returns

Pen width used for painting the scale

See also

[setPenWidth](#), [QwtDial::drawScale\(\)](#)

12.22.4.17 `int QwtRoundScaleDraw::radius () const [inherited]`

Get the radius

Radius is the radius of the backbone without ticks and labels.

See also

[setRadius\(\)](#), [extent\(\)](#)

12.22.4.18 `const QwtScaleDiv & QwtAbstractScaleDraw::scaleDiv () const [inherited]`

Returns

scale division

12.22.4.19 QwtScaleMap & QwtAbstractScaleDraw::scaleMap () [inherited]

Returns

Map how to translate between scale and pixel values

12.22.4.20 void QwtRoundScaleDraw::setAngleRange (double *angle1*, double *angle2*) [inherited]

Adjust the baseline circle segment for round scales.

The baseline will be drawn from min(angle1,angle2) to max(angle1, angle2). The default setting is [-135, 135]. An angle of 0 degrees corresponds to the 12 o'clock position, and positive angles count in a clockwise direction.

Parameters

<i>angle1</i>	
<i>angle2</i>	boundaries of the angle interval in degrees.

Warning

- The angle range is limited to [-360, 360] degrees. Angles exceeding this range will be clipped.
- For angles more than 359 degrees above or below min(angle1, angle2), scale marks will not be drawn.
- If you need a counterclockwise scale, use QwtScaleDiv::setRange

12.22.4.21 void QwtAbstractScaleDraw::setMinimumExtent (int *minExtent*) [inherited]

Set a minimum for the extent.

The extent is calculated from the components of the scale draw. In situations, where the labels are changing and the layout depends on the extent (f.e scrolling a scale), setting an upper limit as minimum extent will avoid jumps of the layout.

Parameters

<i>minExtent</i>	Minimum extent
------------------	----------------

See also

[extent\(\)](#), [minimumExtent\(\)](#)

12.22.4.22 void QwtDialScaleDraw::setPenWidth (uint *penWidth*)

Set the pen width used for painting the scale

Parameters

<i>penWidth</i>	Pen width
-----------------	-----------

See also

[penWidth\(\)](#), [QwtDial::drawScale\(\)](#)

**12.22.4.23 void QwtRoundScaleDraw::setRadius (int *radius*)
[*inherited*]**

Change of radius the scale

Radius is the radius of the backbone without ticks and labels.

Parameters

<i>radius</i>	New Radius
---------------	------------

See also

[moveCenter\(\)](#)

**12.22.4.24 void QwtAbstractScaleDraw::setScaleDiv (const QwtScaleDiv & *sd*)
[*inherited*]**

Change the scale division

Parameters

<i>sd</i>	New scale division
-----------	--------------------

**12.22.4.25 void QwtAbstractScaleDraw::setSpacing (int *spacing*)
[*inherited*]**

Set the spacing between tick and labels.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

[spacing\(\)](#)

12.22.4.26 `void QwtAbstractScaleDraw::setTickLength (
QwtScaleDiv::TickType tickType, int length) [inherited]`

Set the length of the ticks

Parameters

<i>tickType</i>	Tick type
<i>length</i>	New length

Warning

the length is limited to [0..1000]

12.22.4.27 `void QwtAbstractScaleDraw::setTransformation (
QwtScaleTransformation * transformation) [inherited]`

Change the transformation of the scale

Parameters

<i>transformation</i>	New scale transformation
-----------------------	--------------------------

12.22.4.28 `int QwtAbstractScaleDraw::spacing () const [inherited]`

Get the spacing.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

See also

[setSpacing\(\)](#)

12.22.4.29 `const QwtText & QwtAbstractScaleDraw::tickLabel (const QFont
& font, double value) const [protected, inherited]`

Convert a value into its representing label and cache it.

The conversion between value and label is called very often in the layout and painting

code. Unfortunately the calculation of the label sizes might be slow (really slow for rich text in Qt4), so it's necessary to cache the labels.

Parameters

<i>font</i>	Font
<i>value</i>	Value

Returns

Tick label

12.22.4.30 `int QwtAbstractScaleDraw::tickLength (QwtScaleDiv::TickType
tickType) const [inherited]`

Return the length of the ticks

See also

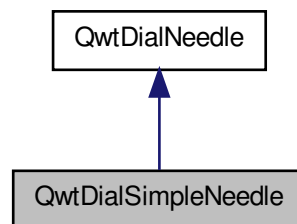
[setTickLength\(\)](#), [majTickLength\(\)](#)

12.23 QwtDialSimpleNeedle Class Reference

A needle for dial widgets.

```
#include <qwt_dial_needle.h>
```

Inheritance diagram for QwtDialSimpleNeedle:



Public Types

- enum [Style](#) {
 Arrow,
 Ray }

Public Member Functions

- virtual void [draw](#) (QPainter *, const QPoint &, int length, double direction, QPalette::ColorGroup=QPalette::Active) const
- const QPalette & [palette](#) () const
- [QwtDialSimpleNeedle](#) ([Style](#), bool hasKnob=true, const QColor &mid=Qt::gray, const QColor &base=Qt::darkGray)
- virtual void [setPalette](#) (const QPalette &)
- void [setWidth](#) (int width)
- int [width](#) () const

Static Public Member Functions

- static void [drawArrowNeedle](#) (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, int width, double direction, bool hasKnob)
- static void [drawRayNeedle](#) (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, int width, double direction, bool hasKnob)

Static Protected Member Functions

- static void [drawKnob](#) (QPainter *, const QPoint &pos, int width, const QBrush &, bool sunken)

12.23.1 Detailed Description

A needle for dial widgets. The following colors are used:

- QColorGroup::Mid
Pointer
- QColorGroup::base
Knob

See also

[QwtDial](#), [QwtCompass](#)

12.23.2 Member Enumeration Documentation**12.23.2.1 enum QwtDialSimpleNeedle::Style**

Style of the needle.

12.23.3 Constructor & Destructor Documentation

12.23.3.1 `QwtDialSimpleNeedle::QwtDialSimpleNeedle (Style style, bool hasKnob = true, const QColor & mid = Qt::gray, const QColor & base = Qt::darkGray)`

Constructor

Parameters

<i>style</i>	Style
<i>hasKnob</i>	With/Without knob
<i>mid</i>	Middle color
<i>base</i>	Base color

12.23.4 Member Function Documentation

12.23.4.1 `void QwtDialSimpleNeedle::draw (QPainter * painter, const QPoint & center, int length, double direction, QPalette::ColorGroup colorGroup = QPalette::Active) const [virtual]`

Draw the needle

Parameters

<i>painter</i>	Painter
<i>center</i>	Center of the dial, start position for the needle
<i>length</i>	Length of the needle
<i>direction</i>	Direction of the needle, in degrees counter clockwise
<i>colorGroup</i>	Color group, used for painting

Implements [QwtDialNeedle](#).

12.23.4.2 `void QwtDialSimpleNeedle::drawArrowNeedle (QPainter * painter, const QPalette & palette, QPalette::ColorGroup colorGroup, const QPoint & center, int length, int width, double direction, bool hasKnob) [static]`

Draw a needle looking like an arrow

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>colorGroup</i>	Color group
<i>center</i>	center of the needle
<i>length</i>	Length of the needle
<i>width</i>	Width of the needle
<i>direction</i>	Current Direction
<i>hasKnob</i>	With/Without knob

12.23.4.3 void QwtDialNeedle::drawKnob (QPainter * *painter*, const QPoint & *pos*, int *width*, const QBrush & *brush*, bool *sunken*) [static, protected, inherited]

Draw the knob.

12.23.4.4 void QwtDialSimpleNeedle::drawRayNeedle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, int *width*, double *direction*, bool *hasKnob*) [static]

Draw a needle looking like a ray

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>colorGroup</i>	Color group
<i>center</i>	center of the needle
<i>length</i>	Length of the needle
<i>width</i>	Width of the needle
<i>direction</i>	Current Direction
<i>hasKnob</i>	With/Without knob

12.23.4.5 const QPalette & QwtDialNeedle::palette () const [inherited]

Returns

the palette of the needle.

12.23.4.6 void QwtDialNeedle::setPalette (const QPalette & *palette*) [virtual, inherited]

Sets the palette for the needle.

Parameters

<i>palette</i>	New Palette
----------------	-------------

12.23.4.7 void QwtDialSimpleNeedle::setWidth (int *width*)

Set the width of the needle

Parameters

<i>width</i>	Width
--------------	-------

See also

[width\(\)](#)

12.23.4.8 int QwtDialSimpleNeedle::width () const**Returns**

the width of the needle

See also

[setWidth\(\)](#)

12.24 QwtDoubleInterval Class Reference

A class representing an interval.

```
#include <qwt_double_interval.h>
```

Public Types

- enum [BorderMode](#) {
IncludeBorders = 0,
ExcludeMinimum = 1,
ExcludeMaximum = 2,
ExcludeBorders = ExcludeMinimum | ExcludeMaximum }

Public Member Functions

- int [borderFlags](#) () const
- bool [contains](#) (double value) const
- [QwtDoubleInterval extend](#) (double value) const
- [QwtDoubleInterval intersect](#) (const [QwtDoubleInterval](#) &) const
- bool [intersects](#) (const [QwtDoubleInterval](#) &) const
- void [invalidate](#) ()
- [QwtDoubleInterval inverted](#) () const
- bool [isNull](#) () const
- bool [isValid](#) () const
- [QwtDoubleInterval limited](#) (double minValue, double maxValue) const
- double [maxValue](#) () const

- double [minValue](#) () const
- [QwtDoubleInterval normalized](#) () const
- int [operator!=](#) (const [QwtDoubleInterval](#) &) const
- [QwtDoubleInterval operator&](#) (const [QwtDoubleInterval](#) &) const
- [QwtDoubleInterval & operator&=](#) (const [QwtDoubleInterval](#) &)
- int [operator==](#) (const [QwtDoubleInterval](#) &) const
- [QwtDoubleInterval operator|](#) (double) const
- [QwtDoubleInterval operator|](#) (const [QwtDoubleInterval](#) &) const
- [QwtDoubleInterval & operator|=](#) (double)
- [QwtDoubleInterval & operator|=](#) (const [QwtDoubleInterval](#) &)
- [QwtDoubleInterval](#) (double min [Value](#), double max [Value](#), int borderFlags=[IncludeBorders](#))
- [QwtDoubleInterval](#) ()
- void [setBorderFlags](#) (int)
- void [setInterval](#) (double min [Value](#), double max [Value](#), int borderFlags=[IncludeBorders](#))
- void [setMax \[Value\]\(#\)](#) (double)
- void [setMin \[Value\]\(#\)](#) (double)
- [QwtDoubleInterval symmetrize](#) (double value) const
- [QwtDoubleInterval unite](#) (const [QwtDoubleInterval](#) &) const
- double [width](#) () const

12.24.1 Detailed Description

A class representing an interval. The interval is represented by 2 doubles, the lower and the upper limit.

12.24.2 Member Enumeration Documentation

12.24.2.1 enum [QwtDoubleInterval::BorderMode](#)

Flag indicating if a border is included/excluded from an interval

- [IncludeBorders](#)
min/max values are inside the interval
- [ExcludeMinimum](#)
min value is not included in the interval
- [ExcludeMaximum](#)
max value is not included in the interval
- [ExcludeBorders](#)
min/max values are not included in the interval

See also

[setBorderMode\(\)](#), [testBorderMode\(\)](#)

12.24.3 Constructor & Destructor Documentation

12.24.3.1 QwtDoubleInterval::QwtDoubleInterval () [inline]

Default Constructor.

Creates an invalid interval [0.0, -1.0]

See also

[setInterval\(\)](#), [isValid\(\)](#)

12.24.3.2 QwtDoubleInterval::QwtDoubleInterval (double *minValue*, double *maxValue*, int *borderFlags* = *IncludeBorders*) [inline]

Constructor

Build an interval with from min/max values

Parameters

<i>minValue</i>	Minimum value
<i>maxValue</i>	Maximum value
<i>borderFlags</i>	Include/Exclude borders

12.24.4 Member Function Documentation

12.24.4.1 int QwtDoubleInterval::borderFlags () const [inline]

Returns

Border flags

See also

[setBorderFlags\(\)](#)

12.24.4.2 bool QwtDoubleInterval::contains (double *value*) const

Test if a value is inside an interval

Parameters

<i>value</i>	Value
--------------	-------

Returns

true, if value \geq `minValue()` && value \leq `maxValue()`

12.24.4.3 QwtDoubleInterval QwtDoubleInterval::extend (double *value*) const

Extend the interval

If value is below minValue, value becomes the lower limit. If value is above maxValue, value becomes the upper limit.

extend has no effect for invalid intervals

Parameters

<i>value</i>	Value
--------------	-------

See also

[isValid\(\)](#)

12.24.4.4 QwtDoubleInterval QwtDoubleInterval::intersect (const QwtDoubleInterval & *other*) const

Intersect 2 intervals.

12.24.4.5 bool QwtDoubleInterval::intersects (const QwtDoubleInterval & *other*) const

Test if two intervals overlap

12.24.4.6 void QwtDoubleInterval::invalidate () [inline]

Invalidate the interval

The limits are set to interval [0.0, -1.0]

See also

[isValid\(\)](#)

12.24.4.7 QwtDoubleInterval QwtDoubleInterval::inverted () const

Invert the limits of the interval

Returns

Inverted interval

See also

[normalized\(\)](#)

12.24.4.8 bool QwtDoubleInterval::isNull () const [inline]**Returns**

true, if [isValid\(\)](#) && ([minValue\(\)](#) >= [maxValue\(\)](#))

12.24.4.9 bool QwtDoubleInterval::isValid () const [inline]

A interval is valid when [minValue\(\)](#) <= [maxValue\(\)](#). In case of [QwtDoubleInterval::ExcludeBorders](#) it is true when [minValue\(\)](#) < [maxValue\(\)](#)

12.24.4.10 QwtDoubleInterval QwtDoubleInterval::limited (double *lowerBound*, double *upperBound*) const

Limit the interval, keeping the border modes

Parameters

<i>lowerBound</i>	Lower limit
<i>upperBound</i>	Upper limit

Returns

Limited interval

12.24.4.11 double QwtDoubleInterval::maxValue () const [inline]**Returns**

Upper limit of the interval

12.24.4.12 double QwtDoubleInterval::minValue () const [inline]

Returns

Lower limit of the interval

12.24.4.13 QwtDoubleInterval QwtDoubleInterval::normalized () const

Normalize the limits of the interval.

If [maxValue\(\)](#) < [minValue\(\)](#) the limits will be inverted.

Returns

Normalized interval

See also

[isValid\(\)](#), [inverted\(\)](#)

12.24.4.14 int QwtDoubleInterval::operator!= (const QwtDoubleInterval & *other*) const [inline]

Compare two intervals.

12.24.4.15 QwtDoubleInterval QwtDoubleInterval::operator& (const QwtDoubleInterval & *interval*) const [inline]

Intersection of two intervals

See also

[intersect\(\)](#)

12.24.4.16 QwtDoubleInterval & QwtDoubleInterval::operator&= (const QwtDoubleInterval & *interval*)

Intersects this interval with the given interval.

12.24.4.17 int QwtDoubleInterval::operator== (const QwtDoubleInterval & *other*) const [inline]

Compare two intervals.

12.24.4.18 `QwtDoubleInterval QwtDoubleInterval::operator| (const
QwtDoubleInterval & interval) const [inline]`

Union of two intervals

See also

[unite\(\)](#)

12.24.4.19 `QwtDoubleInterval QwtDoubleInterval::operator| (double value)
const [inline]`

Extend an interval

See also

[extend\(\)](#)

12.24.4.20 `QwtDoubleInterval & QwtDoubleInterval::operator|= (const
QwtDoubleInterval & interval)`

Unites this interval with the given interval.

12.24.4.21 `void QwtDoubleInterval::setBorderFlags (int borderFlags)
[inline]`

Change the border flags

Parameters

<i>borderFlags</i>	Or'd BorderMode flags
--------------------	-----------------------

See also

[borderFlags\(\)](#)

12.24.4.22 `void QwtDoubleInterval::setInterval (double minValue, double
maxValue, int borderFlags = IncludeBorders) [inline]`

Assign the limits of the interval

Parameters

<i>minValue</i>	Minimum value
<i>maxValue</i>	Maximum value
<i>borderFlags</i>	Include/Exclude borders

**12.24.4.23 void QwtDoubleInterval::setMaxValue (double *maxValue*)
[inline]**

Assign the upper limit of the interval

Parameters

<i>maxValue</i>	Maximum value
-----------------	---------------

**12.24.4.24 void QwtDoubleInterval::setMinValue (double *minValue*)
[inline]**

Assign the lower limit of the interval

Parameters

<i>minValue</i>	Minimum value
-----------------	---------------

12.24.4.25 QwtDoubleInterval QwtDoubleInterval::symmetrize (double *value*) const

Adjust the limit that is closer to value, so that value becomes the center of the interval.

Parameters

<i>value</i>	Center
--------------	--------

Returns

Interval with value as center

**12.24.4.26 QwtDoubleInterval QwtDoubleInterval::unite (const
QwtDoubleInterval & *other*) const**

Unite 2 intervals.

12.24.4.27 double QwtDoubleInterval::width () const [inline]

Return the width of an interval The width of invalid intervals is 0.0, otherwise the result is [maxValue\(\)](#) - [minValue\(\)](#).

See also

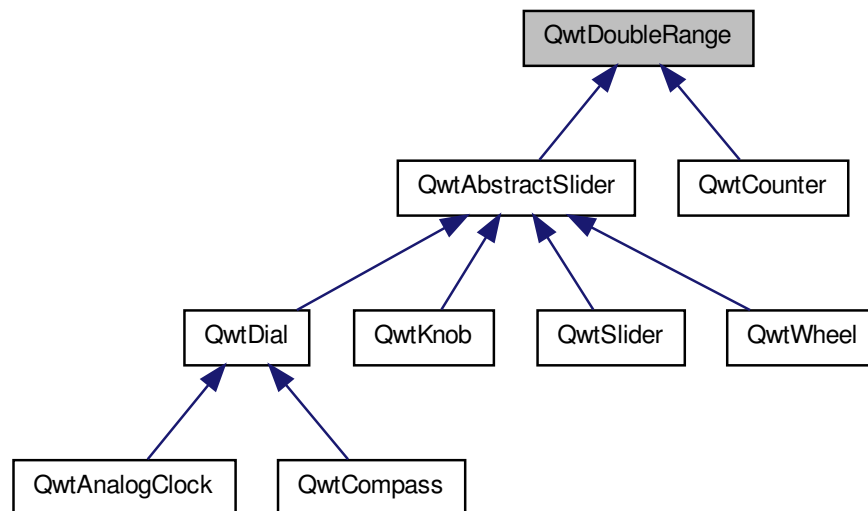
[isValid\(\)](#)

12.25 QwtDoubleRange Class Reference

A class which controls a value within an interval.

```
#include <qwt_double_range.h>
```

Inheritance diagram for QwtDoubleRange:



Public Member Functions

- virtual void [fitValue](#) (double)
- virtual void [incPages](#) (int)
- virtual void [incValue](#) (int)
- bool [isValid](#) () const
- double [maxValue](#) () const
- double [minValue](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtDoubleRange](#) ()
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setStep](#) (double)
- void [setValid](#) (bool)
- virtual void [setValue](#) (double)
- double [step](#) () const

- double [value](#) () const
- virtual [~QwtDoubleRange](#) ()

Protected Member Functions

- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- virtual void [stepChange](#) ()
- virtual void [valueChange](#) ()

12.25.1 Detailed Description

A class which controls a value within an interval. This class is useful as a base class or a member for sliders. It represents an interval of type double within which a value can be moved. The value can be either an arbitrary point inside the interval (see [QwtDoubleRange::setValue](#)), or it can be fitted into a step raster (see [QwtDoubleRange::fitValue](#) and [QwtDoubleRange::incValue](#)).

As a special case, a [QwtDoubleRange](#) can be periodic, which means that a value outside the interval will be mapped to a value inside the interval when [QwtDoubleRange::setValue\(\)](#), [QwtDoubleRange::fitValue\(\)](#), [QwtDoubleRange::incValue\(\)](#) or [QwtDoubleRange::incPages\(\)](#) are called.

12.25.2 Constructor & Destructor Documentation

12.25.2.1 QwtDoubleRange::QwtDoubleRange ()

The range is initialized to [0.0, 100.0], the step size to 1.0, and the value to 0.0.

12.25.2.2 QwtDoubleRange::~~QwtDoubleRange () [virtual]

Destroys the [QwtDoubleRange](#).

12.25.3 Member Function Documentation

12.25.3.1 double QwtDoubleRange::exactPrevValue () const [protected]

Returns the exact previous value.

12.25.3.2 double QwtDoubleRange::exactValue () const [protected]

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.25.3.3 void QwtDoubleRange::fitValue (double *x*) [virtual]

Adjust the value to the closest point in the step raster.

Parameters

<i>x</i> value

Warning

The value is clipped when it lies outside the range. When the range is [QwtDoubleRange::periodic](#), it will be mapped to a point in the interval such that

$\text{new value} := x + n * (\text{max. value} - \text{min. value})$

with an integer number *n*.

Reimplemented in [QwtAbstractSlider](#).

12.25.3.4 void QwtDoubleRange::incPages (int *nPages*) [virtual]

Increment the value by a specified number of pages.

Parameters

<i>nPages</i> Number of pages to increment. A negative number decrements the value.

Warning

The Page size is specified in the constructor.

12.25.3.5 void QwtDoubleRange::incValue (int *nSteps*) [virtual]

Increment the value by a specified number of steps.

Parameters

<i>nSteps</i> Number of steps to increment
--

Warning

As a result of this operation, the new value will always be adjusted to the step raster.

Reimplemented in [QwtAbstractSlider](#).

12.25.3.6 bool QwtDoubleRange::isValid () const

Indicates if the value is valid.

Reimplemented in [QwtAbstractSlider](#).

12.25.3.7 double QwtDoubleRange::maxValue () const

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.25.3.8 double QwtDoubleRange::minValue () const

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.25.3.9 int QwtDoubleRange::pageSize () const

Returns the page size in steps.

12.25.3.10 `bool QwtDoubleRange::periodic () const`

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.25.3.11 `double QwtDoubleRange::prevValue () const` **[protected]**

Returns the previous value.

12.25.3.12 `void QwtDoubleRange::rangeChange ()` **[protected, virtual]**

Notify a change of the range.

This virtual function is called whenever the range changes. The default implementation does nothing.

Reimplemented in [QwtCounter](#), [QwtDial](#), and [QwtSlider](#).

12.25.3.13 `void QwtDoubleRange::setPeriodic (bool tf)`

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

$$\text{point} = \text{value} + n * \text{width}$$

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i> true for a periodic range

12.25.3.14 `void QwtDoubleRange::setRange (double vmin, double vmax, double vstep = 0.0, int pageSize = 1)`

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will **not** be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.25.3.15 void QwtDoubleRange::setStep (double *vstep*)

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

12.25.3.16 void QwtDoubleRange::setValid (bool *isValid*)

Set the value to be valid/invalid.

Reimplemented in [QwtAbstractSlider](#).

12.25.3.17 void QwtDoubleRange::setValue (double *x*) [virtual]

Set a new value without adjusting to the step raster.

Parameters

x new value

Warning

The value is clipped when it lies outside the range. When the range is [QwtDoubleRange::periodic](#), it will be mapped to a point in the interval such that

$$\text{new value} := x + n * (\text{max. value} - \text{min. value})$$

with an integer number n .

Reimplemented in [QwtAbstractSlider](#), and [QwtCounter](#).

12.25.3.18 double QwtDoubleRange::step () const**Returns**

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.25.3.19 void QwtDoubleRange::stepChange () [protected, virtual]

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.25.3.20 double QwtDoubleRange::value () const

Returns the current value.

Reimplemented in [QwtCounter](#).

12.25.3.21 void QwtDoubleRange::valueChange () [protected, virtual]

Notify a change of value.

This virtual function is called whenever the value changes. The default implementation does nothing.

Reimplemented in [QwtAbstractSlider](#), [QwtDial](#), [QwtSlider](#), and [QwtWheel](#).

12.26 QwtDynGridLayout Class Reference

The [QwtDynGridLayout](#) class lays out widgets in a grid, adjusting the number of columns and rows to the current size.

```
#include <qwt_dyngrid_layout.h>
```

Public Member Functions

- virtual void [addItem](#) (QLayoutItem *)
- virtual uint [columnsForWidth](#) (int width) const
- virtual int [count](#) () const
- virtual Qt::Orientations [expandingDirections](#) () const
- virtual bool [hasHeightForWidth](#) () const
- virtual int [heightForWidth](#) (int) const
- virtual void [invalidate](#) ()
- virtual bool [isEmpty](#) () const
- virtual QLayoutItem * [itemAt](#) (int index) const
- uint [itemCount](#) () const
- QList< QRect > [layoutItems](#) (const QRect &, uint numCols) const
- uint [maxCols](#) () const
- virtual int [maxItemWidth](#) () const
- uint [numCols](#) () const
- uint [numRows](#) () const
- [QwtDynGridLayout](#) (QWidget *, int margin=0, int space=-1)
- [QwtDynGridLayout](#) (int space=-1)
- void [setExpandingDirections](#) (Qt::Orientations)
- virtual void [setGeometry](#) (const QRect &rect)
- void [setMaxCols](#) (uint maxCols)
- virtual QSize [sizeHint](#) () const
- virtual QLayoutItem * [takeAt](#) (int index)
- virtual ~[QwtDynGridLayout](#) ()

Protected Member Functions

- void [layoutGrid](#) (uint numCols, QwtArray< int > &rowHeight, QwtArray< int > &colWidth) const
- void [stretchGrid](#) (const QRect &rect, uint numCols, QwtArray< int > &rowHeight, QwtArray< int > &colWidth) const

12.26.1 Detailed Description

The [QwtDynGridLayout](#) class lays out widgets in a grid, adjusting the number of columns and rows to the current size. [QwtDynGridLayout](#) takes the space it gets, divides it up into rows and columns, and puts each of the widgets it manages into the correct cell(s). It lays out as many number of columns as possible (limited by [maxCols\(\)](#)).

12.26.2 Constructor & Destructor Documentation

12.26.2.1 QwtDynGridLayout::QwtDynGridLayout (QWidget * *parent*, int *margin* = 0, int *spacing* = -1) [explicit]

Parameters

<i>parent</i>	Parent widget
<i>margin</i>	Margin
<i>spacing</i>	Spacing

12.26.2.2 QwtDynGridLayout::QwtDynGridLayout (int *spacing* = -1) [explicit]

Parameters

<i>spacing</i>	Spacing
----------------	---------

12.26.2.3 QwtDynGridLayout::~QwtDynGridLayout () [virtual]

Destructor.

12.26.3 Member Function Documentation

12.26.3.1 void QwtDynGridLayout::addItem (QLayoutItem * *item*) [virtual]

Adds item to the next free position.

12.26.3.2 `uint QwtDynGridLayout::columnsForWidth (int width) const` `[virtual]`

Calculate the number of columns for a given width. It tries to use as many columns as possible (limited by [maxCols\(\)](#))

Parameters

<i>width</i>	Available width for all columns
--------------	---------------------------------

See also

[maxCols\(\)](#), [setMaxCols\(\)](#)

12.26.3.3 `int QwtDynGridLayout::count () const` `[virtual]`

Returns

Number of items in the layout

12.26.3.4 `Qt::Orientations QwtDynGridLayout::expandingDirections ()` `const [virtual]`

Returns whether this layout can make use of more space than [sizeHint\(\)](#). A value of `Qt::Vertical` or `Qt::Horizontal` means that it wants to grow in only one dimension, while `Qt::Vertical | Qt::Horizontal` means that it wants to grow in both dimensions.

See also

[setExpandingDirections\(\)](#)

12.26.3.5 `bool QwtDynGridLayout::hasHeightForWidth () const` `[virtual]`

Returns

true: [QwtDynGridLayout](#) implements `heightForWidth`.

See also

[heightForWidth\(\)](#)

12.26.3.6 `int QwtDynGridLayout::heightForWidth (int width) const`
[**virtual**]

Returns

The preferred height for this layout, given the width *w*.

See also

[hasHeightForWidth\(\)](#)

12.26.3.7 `void QwtDynGridLayout::invalidate ()` [**virtual**]

Invalidate all internal caches.

12.26.3.8 `bool QwtDynGridLayout::isEmpty () const` [**virtual**]

Returns

true if this layout is empty.

12.26.3.9 `QLayoutItem * QwtDynGridLayout::itemAt (int index) const`
[**virtual**]

Find the item at a specific index

Parameters

<i>index</i>	Index
--------------	-------

See also

[takeAt\(\)](#)

12.26.3.10 `uint QwtDynGridLayout::itemCount () const`

Returns

number of layout items

12.26.3.11 `void QwtDynGridLayout::layoutGrid (uint numCols, QwtArray< int > & rowHeight, QwtArray< int > & colWidth) const`
[protected]

Calculate the dimensions for the columns and rows for a grid of *numCols* columns.

Parameters

<i>numCols</i>	Number of columns.
<i>rowHeight</i>	Array where to fill in the calculated row heights.
<i>colWidth</i>	Array where to fill in the calculated column widths.

12.26.3.12 `QList< QRect > QwtDynGridLayout::layoutItems (const QRect & rect, uint numCols) const`

Calculate the geometries of the layout items for a layout with *numCols* columns and a given *rect*.

Parameters

<i>rect</i>	Rect where to place the items
<i>numCols</i>	Number of columns

Returns

item geometries

12.26.3.13 `uint QwtDynGridLayout::maxCols () const`

Return the upper limit for the number of columns. 0 means unlimited, what is the default.

See also

[setMaxCols\(\)](#)

12.26.3.14 `int QwtDynGridLayout::maxItemWidth () const` **[virtual]**

Returns

the maximum width of all layout items

12.26.3.15 `uint QwtDynGridLayout::numCols () const`

Returns

Number of columns of the current layout.

See also

[numRows\(\)](#)

Warning

The number of columns might change whenever the geometry changes

12.26.3.16 uint QwtDynGridLayout::numRows () const**Returns**

Number of rows of the current layout.

See also

[numCols\(\)](#)

Warning

The number of rows might change whenever the geometry changes

12.26.3.17 void QwtDynGridLayout::setExpandingDirections (Qt::Orientations *expanding*)

Set whether this layout can make use of more space than [sizeHint\(\)](#). A value of Qt::Vertical or Qt::Horizontal means that it wants to grow in only one dimension, while Qt::Vertical | Qt::Horizontal means that it wants to grow in both dimensions. The default value is 0.

Parameters

<i>expanding</i> Or'd orientations

See also

[expandingDirections\(\)](#)

12.26.3.18 void QwtDynGridLayout::setGeometry (const QRect & *rect*) [virtual]

Reorganizes columns and rows and resizes managed widgets within the rectangle *rect*.

Parameters

<i>rect</i>	Layout geometry
-------------	-----------------

12.26.3.19 void QwtDynGridLayout::setMaxCols (uint *maxCols*)

Limit the number of columns.

Parameters

<i>maxCols</i>	upper limit, 0 means unlimited
----------------	--------------------------------

See also

[maxCols\(\)](#)

12.26.3.20 QSize QwtDynGridLayout::sizeHint () const [virtual]

Return the size hint. If [maxCols\(\)](#) > 0 it is the size for a grid with [maxCols\(\)](#) columns, otherwise it is the size for a grid with only one row.

See also

[maxCols\(\)](#), [setMaxCols\(\)](#)

12.26.3.21 void QwtDynGridLayout::stretchGrid (const QRect & *rect*, uint *numCols*, QwtArray< int > & *rowHeight*, QwtArray< int > & *colWidth*) const [protected]

Stretch columns in case of [expanding\(\)](#) & [QSizePolicy::Horizontal](#) and rows in case of [expanding\(\)](#) & [QSizePolicy::Vertical](#) to fill the entire rect. Rows and columns are stretched with the same factor.

See also

[setExpanding\(\)](#), [expanding\(\)](#)

12.26.3.22 QLayoutItem * QwtDynGridLayout::takeAt (int *index*) [virtual]

Find the item at a specific index and remove it from the layout

Parameters

<i>index</i>	Index
--------------	-------

See also

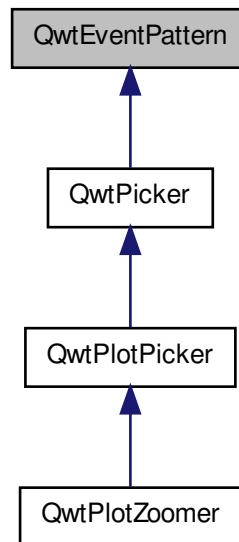
[itemAt\(\)](#)

12.27 QwtEventPattern Class Reference

A collection of event patterns.

```
#include <qwt_event_pattern.h>
```

Inheritance diagram for QwtEventPattern:



Classes

- class [KeyPattern](#)
A pattern for key events.
- class [MousePattern](#)
A pattern for mouse events.

Public Types

- enum [KeyPatternCode](#) {

```

    KeySelect1,
    KeySelect2,
    KeyAbort,
    KeyLeft,
    KeyRight,
    KeyUp,
    KeyDown,
    KeyRedo,
    KeyUndo,
    KeyHome,
    KeyPatternCount }
• enum MousePatternCode {
    MouseSelect1,
    MouseSelect2,
    MouseSelect3,
    MouseSelect4,
    MouseSelect5,
    MouseSelect6,
    MousePatternCount }

```

Public Member Functions

- void [initKeyPattern](#) ()
- void [initMousePattern](#) (int numButtons)
- bool [keyMatch](#) (uint pattern, const QKeyEvent *) const
- const QwtArray< [KeyPattern](#) > & [keyPattern](#) () const
- QwtArray< [KeyPattern](#) > & [keyPattern](#) ()
- bool [mouseMatch](#) (uint pattern, const QMouseEvent *) const
- const QwtArray< [MousePattern](#) > & [mousePattern](#) () const
- QwtArray< [MousePattern](#) > & [mousePattern](#) ()
- [QwtEventPattern](#) ()
- void [setKeyPattern](#) (uint pattern, int key, int state=Qt::NoButton)
- void [setKeyPattern](#) (const QwtArray< [KeyPattern](#) > &)
- void [setMousePattern](#) (const QwtArray< [MousePattern](#) > &)
- void [setMousePattern](#) (uint pattern, int button, int state=Qt::NoButton)
- virtual [~QwtEventPattern](#) ()

Protected Member Functions

- virtual bool [keyMatch](#) (const [KeyPattern](#) &, const QKeyEvent *) const
- virtual bool [mouseMatch](#) (const [MousePattern](#) &, const QMouseEvent *) const

12.27.1 Detailed Description

A collection of event patterns. [QwtEventPattern](#) introduces an level of indirection for mouse and keyboard inputs. Those are represented by symbolic names, so the application code can be configured by individual mappings.

See also

[QwtPicker](#), [QwtPickerMachine](#), [QwtPlotZoomer](#)

12.27.2 Member Enumeration Documentation

12.27.2.1 enum QwtEventPattern::KeyPatternCode

Symbolic keyboard input codes.

Default initialization:

- KeySelect1
Qt::Key_Return
- KeySelect2
Qt::Key_Space
- KeyAbort
Qt::Key_Escape
- KeyLeft
Qt::Key_Left
- KeyRight
Qt::Key_Right
- KeyUp
Qt::Key_Up
- KeyDown
Qt::Key_Down
- KeyUndo
Qt::Key_Minus
- KeyRedo
Qt::Key_Plus
- KeyHome
Qt::Key_Escape

12.27.2.2 enum QwtEventPattern::MousePatternCode

Symbolic mouse input codes.

The default initialization for 3 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::MidButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::MidButton + Qt::ShiftButton

The default initialization for 2 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

The default initialization for 1 button mice is:

- MouseSelect1
Qt::LeftButton

- MouseSelect2
Qt::LeftButton + Qt::ControlButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::LeftButton + Qt::ControlButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

See also

[initMousePattern\(\)](#)

12.27.3 Constructor & Destructor Documentation

12.27.3.1 QwtEventPattern::QwtEventPattern ()

Constructor

See also

[MousePatternCode](#), [KeyPatternCode](#)

12.27.3.2 QwtEventPattern::~QwtEventPattern () [virtual]

Destructor.

12.27.4 Member Function Documentation

12.27.4.1 void QwtEventPattern::initKeyPattern ()

Set default mouse patterns.

See also

[KeyPatternCode](#)

12.27.4.2 void QwtEventPattern::initMousePattern (int *numButtons*)

Set default mouse patterns, depending on the number of mouse buttons

Parameters

<i>numButtons</i>	Number of mouse buttons (≤ 3)
-------------------	--------------------------------------

See also

[MousePatternCode](#)

12.27.4.3 bool QwtEventPattern::keyMatch (uint *pattern*, const QKeyEvent * *e*) const

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.27.4.4 bool QwtEventPattern::keyMatch (const KeyPattern & *pattern*, const QKeyEvent * *e*) const [protected, virtual]

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Key event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.27.4.5 `const QwtArray< QwtEventPattern::KeyPattern > & QwtEventPattern::keyPattern () const`

Return key patterns.

12.27.4.6 `QwtArray< QwtEventPattern::KeyPattern > & QwtEventPattern::keyPattern ()`

Return Key patterns.

12.27.4.7 `bool QwtEventPattern::mouseMatch (uint pattern, const QMouseEvent * e) const`

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.27.4.8 `bool QwtEventPattern::mouseMatch (const MousePattern & pattern, const QMouseEvent * e) const` **[protected, virtual]**

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Mouse event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.27.4.9 QwtArray< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern ()

Return ,ouse patterns.

12.27.4.10 const QwtArray< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern () const

Return mouse patterns.

12.27.4.11 void QwtEventPattern::setKeyPattern (uint *pattern*, int *key*, int *state* = `Qt::NoButton`)

Change one key pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>key</i>	Key
<i>state</i>	State

See also

[QKeyEvent](#)

12.27.4.12 void QwtEventPattern::setKeyPattern (const QwtArray< KeyPattern > & *pattern*)

Change the key event patterns.

12.27.4.13 `void QwtEventPattern::setMousePattern (uint pattern, int button,
int state = Qt::NoButton)`

Change one mouse pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>button</i>	Button
<i>state</i>	State

See also

QMouseEvent

12.27.4.14 `void QwtEventPattern::setMousePattern (const QwtArray<
MousePattern > & pattern)`

Change the mouse event patterns.

12.28 QwtIntervalData Class Reference

Series of samples of a value and an interval.

```
#include <qwt_interval_data.h>
```

Public Member Functions

- QwtDoubleRect [boundingRect](#) () const
- const [QwtDoubleInterval](#) & [interval](#) (size_t i) const
- [QwtIntervalData](#) (const QwtArray< [QwtDoubleInterval](#) > &, const QwtArray< double > &)
- [QwtIntervalData](#) ()
- void [setData](#) (const QwtArray< [QwtDoubleInterval](#) > &, const QwtArray< double > &)
- size_t [size](#) () const
- double [value](#) (size_t i) const
- [~QwtIntervalData](#) ()

12.28.1 Detailed Description

Series of samples of a value and an interval. [QwtIntervalData](#) is a series of samples of a value and an interval. F.e. error bars are built from samples [x, y1-y2], while a histogram might consist of [x1-x2, y] samples.

12.28.2 Constructor & Destructor Documentation

12.28.2.1 QwtIntervalData::QwtIntervalData ()

Constructor.

12.28.2.2 QwtIntervalData::QwtIntervalData (const QwtArray< QwtDoubleInterval > & *intervals*, const QwtArray< double > & *values*)

Constructor.

12.28.2.3 QwtIntervalData::~QwtIntervalData ()

Destructor.

12.28.3 Member Function Documentation

12.28.3.1 QwtDoubleRect QwtIntervalData::boundingRect () const

Calculate the bounding rectangle of the samples

The x coordinates of the rectangle are built from the intervals, the y coordinates from the values.

Returns

Bounding rectangle

**12.28.3.2 const QwtDoubleInterval & QwtIntervalData::interval (size_t *i*)
const [inline]**

Interval of a sample

Parameters

<i>i</i>	Sample index
----------	--------------

Returns

Interval

See also[value\(\)](#), [size\(\)](#)

12.28.3.3 `void QwtIntervalData::setData (const QwtArray< QwtDoubleInterval > & intervals, const QwtArray< double > & values)`

Assign samples.

12.28.3.4 `size_t QwtIntervalData::size () const [inline]`

Returns

Number of samples

12.28.3.5 `double QwtIntervalData::value (size_t i) const [inline]`

Value of a sample

Parameters

<i>i</i>	Sample index
----------	--------------

Returns

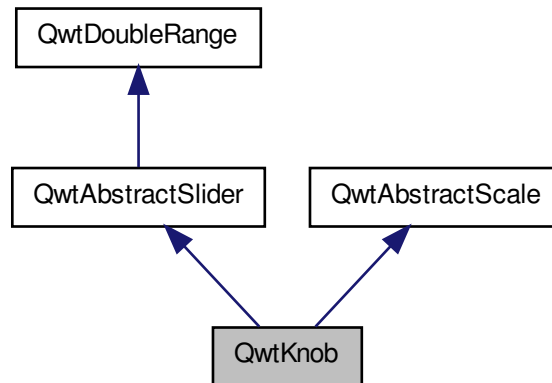
Value

See also[interval\(\)](#), [size\(\)](#)**12.29 QwtKnob Class Reference**

The Knob Widget.

```
#include <qwt_knob.h>
```

Inheritance diagram for QwtKnob:



Public Types

- enum [ScrollMode](#) {
 ScrNone,
 ScrMouse,
 ScrTimer,
 ScrDirect,
 ScrPage }
- enum [Symbol](#) {
 Line,
 Dot }

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- virtual void [setReadOnly](#) (bool)
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()

- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- bool [autoScale](#) () const
- int [borderWidth](#) () const
- virtual void [incPages](#) (int)
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- int [knobWidth](#) () const
- virtual double [mass](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const
- double [minValue](#) () const
- Qt::Orientation [orientation](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtKnob](#) (QWidget *parent=NULL)
- const [QwtRoundScaleDraw](#) * [scaleDraw](#) () const
- [QwtRoundScaleDraw](#) * [scaleDraw](#) ()
- [QwtScaleEngine](#) * [scaleEngine](#) ()
- const [QwtScaleEngine](#) * [scaleEngine](#) () const
- const [QwtScaleMap](#) & [scaleMap](#) () const
- int [scaleMaxMajor](#) () const
- int [scaleMaxMinor](#) () const
- void [setAutoScale](#) ()
- void [setBorderWidth](#) (int bw)
- void [setKnobWidth](#) (int w)
- virtual void [setMass](#) (double val)
- virtual void [setOrientation](#) (Qt::Orientation o)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setScale](#) (const [QwtScaleDiv](#) &s)
- void [setScale](#) (double vmin, double vmax, double step=0.0)
- void [setScale](#) (const [QwtDoubleInterval](#) &, double step=0.0)
- void [setScaleDraw](#) ([QwtRoundScaleDraw](#) *)
- void [setScaleEngine](#) ([QwtScaleEngine](#) *)
- void [setScaleMaxMajor](#) (int ticks)
- void [setScaleMaxMinor](#) (int ticks)
- void [setStep](#) (double)
- void [setSymbol](#) ([Symbol](#))
- void [setTotalAngle](#) (double angle)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)

- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- void [stopMoving](#) ()
- [Symbol](#) [symbol](#) () const
- double [totalAngle](#) () const
- double [value](#) () const
- virtual [~QwtKnob](#) ()

Protected Member Functions

- const [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) () const
- [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) ()
- void [draw](#) (QPainter *p, const QRect &r)
- void [drawKnob](#) (QPainter *p, const QRect &r)
- void [drawMarker](#) (QPainter *p, double arc, const QColor &c)
- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [keyPressEvent](#) (QKeyEvent *e)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)
- double [mouseOffset](#) () const
- virtual void [mousePressEvent](#) (QMouseEvent *e)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *e)
- virtual void [paintEvent](#) (QPaintEvent *e)
- double [prevValue](#) () const
- void [rescale](#) (double vmin, double vmax, double step=0.0)
- virtual void [resizeEvent](#) (QResizeEvent *e)
- int [scrollMode](#) () const
- void [setAbstractScaleDraw](#) ([QwtAbstractScaleDraw](#) *)
- void [setMouseOffset](#) (double)
- virtual void [setPosition](#) (const QPoint &)
- virtual void [stepChange](#) ()
- virtual void [timerEvent](#) (QTimerEvent *e)
- virtual void [wheelEvent](#) (QWheelEvent *e)

12.29.1 Detailed Description

The Knob Widget. The [QwtKnob](#) widget imitates look and behaviour of a volume knob on a radio. It contains a scale around the knob which is set up automatically or can be configured manually (see [QwtAbstractScale](#)). Automatic scrolling is enabled when the user presses a mouse button on the scale. For a description of signals, slots and other members, see [QwtAbstractSlider](#).

See also

[QwtAbstractSlider](#) and [QwtAbstractScale](#) for the descriptions of the inherited members.

12.29.2 Member Enumeration Documentation

12.29.2.1 enum QwtAbstractSlider::ScrollMode [inherited]

Scroll mode

See also

[getScrollMode\(\)](#)

12.29.2.2 enum QwtKnob::Symbol

Symbol

See also

[QwtKnob::QwtKnob\(\)](#)

12.29.3 Constructor & Destructor Documentation

12.29.3.1 QwtKnob::QwtKnob (QWidget * *parent* = *NULL*) [explicit]

Constructor

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.29.3.2 QwtKnob::~~QwtKnob () [virtual]

Destructor.

12.29.4 Member Function Documentation

12.29.4.1 const QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () const [protected, inherited]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.29.4.2 QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () [protected, inherited]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.29.4.3 bool QwtAbstractScale::autoScale () const [inherited]

Returns

`true` if autoscaling is enabled

12.29.4.4 int QwtKnob::borderWidth () const

Return the border width.

12.29.4.5 void QwtKnob::draw (QPainter * *painter*, const QRect & *rect*) [protected]

Repaint the knob

Parameters

<i>painter</i>	Painter
<i>rect</i>	Update rectangle

12.29.4.6 void QwtKnob::drawKnob (QPainter * *painter*, const QRect & *r*) [protected]

Draw the knob.

Parameters

<i>painter</i>	painter
<i>r</i>	Bounding rectangle of the knob (without scale)

12.29.4.7 `void QwtKnob::drawMarker (QPainter * p, double arc, const QColor & c)` **[protected]**

Draw the marker at the knob's front.

Parameters

<i>p</i>	Painter
<i>arc</i>	Angle of the marker
<i>c</i>	Marker color

12.29.4.8 `double QwtDoubleRange::exactPrevValue () const` **[protected, inherited]**

Returns the exact previous value.

12.29.4.9 `double QwtDoubleRange::exactValue () const` **[protected, inherited]**

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.29.4.10 `void QwtAbstractSlider::fitValue (double value)` **[virtual, slot, inherited]**

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.29.4.11 void QwtDoubleRange::incPages (int *nPages*) [virtual, inherited]

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.29.4.12 void QwtAbstractSlider::incValue (int *steps*) [virtual, slot, inherited]

Increment the value by a specified number of steps.

Parameters

<i>steps</i>	number of steps
--------------	-----------------

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.29.4.13 bool QwtAbstractSlider::isReadOnly () const [inherited]

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.29.4.14 bool QwtAbstractSlider::isValid () const [inline, inherited]

See also

[QwtDbfRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.29.4.15 `void QwtAbstractSlider::keyPressEvent (QKeyEvent * e)`
[protected, virtual, inherited]

Handles key events

- Key_Down, Key_Left
Decrement by 1
- Key_Up, Key_Right
Increment by 1

Parameters

<i>e</i>	Key event
----------	-----------

See also

[isReadOnly\(\)](#)

Reimplemented in [QwtCompass](#), and [QwtDial](#).

12.29.4.16 `int QwtKnob::knobWidth () const`

Return the width of the knob.

12.29.4.17 `double QwtAbstractSlider::mass () const` **[virtual, inherited]**

Returns

mass

See also

[setMass\(\)](#)

Reimplemented in [QwtWheel](#).

12.29.4.18 double QwtDoubleRange::maxValue () const [inherited]

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.29.4.19 QSize QwtKnob::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

The return value of [QwtKnob::minimumSizeHint\(\)](#) depends on the font and the scale.

12.29.4.20 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.29.4.21 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * *e*) [protected, virtual, inherited]

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.29.4.22 `void QwtAbstractSlider::mousePressEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.29.4.23 `void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.29.4.24 `Qt::Orientation QwtAbstractSlider::orientation () const`
[inherited]

Returns

Orientation

See also

[setOrientation\(\)](#)

12.29.4.25 `int QwtDoubleRange::pageSize () const` [inherited]

Returns the page size in steps.

12.29.4.26 `void QwtKnob::paintEvent (QPaintEvent * e)` [protected, virtual]

Repaint the knob

Parameters

<i>e</i>	Paint event
----------	-------------

12.29.4.27 `bool QwtDoubleRange::periodic () const [inherited]`

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.29.4.28 `double QwtDoubleRange::prevValue () const [protected, inherited]`

Returns the previous value.

12.29.4.29 `void QwtAbstractScale::rescale (double vmin, double vmax, double stepSize = 0.0) [protected, inherited]`

Recalculate the scale division and update the scale draw.

Parameters

<i>vmin</i>	Lower limit of the scale interval
<i>vmax</i>	Upper limit of the scale interval
<i>stepSize</i>	Major step size

See also

[scaleChange\(\)](#)

12.29.4.30 `void QwtKnob::resizeEvent (QResizeEvent * e) [protected, virtual]`

Qt Resize Event

12.29.4.31 `const QwtRoundScaleDraw * QwtKnob::scaleDraw () const`

Returns

the scale draw of the knob

See also

[setScaleDraw\(\)](#)

12.29.4.32 QwtRoundScaleDraw * QwtKnob::scaleDraw ()**Returns**

the scale draw of the knob

See also

[setScaleDraw\(\)](#)

12.29.4.33 const QwtScaleEngine * QwtAbstractScale::scaleEngine () const
[inherited]**Returns**

Scale engine

See also

[setScaleEngine\(\)](#)

12.29.4.34 QwtScaleEngine * QwtAbstractScale::scaleEngine ()
[inherited]**Returns**

Scale engine

See also

[setScaleEngine\(\)](#)

12.29.4.35 const QwtScaleMap & QwtAbstractScale::scaleMap () const
[inherited]**Returns**

[abstractScaleDraw\(\)->scaleMap\(\)](#)

12.29.4.36 `int QwtAbstractScale::scaleMaxMajor () const [inherited]`

Returns

Max. number of major tick intervals The default value is 5.

12.29.4.37 `int QwtAbstractScale::scaleMaxMinor () const [inherited]`

Returns

Max. number of minor tick intervals The default value is 3.

12.29.4.38 `void QwtAbstractScale::setAbstractScaleDraw (
QwtAbstractScaleDraw * scaleDraw) [protected,
inherited]`

Set a scale draw.

scaleDraw has to be created with `new` and will be deleted in `~QwtAbstractScale` or the next call of `setAbstractScaleDraw`.

12.29.4.39 `void QwtAbstractScale::setAutoScale () [inherited]`

Advise the widget to control the scale range internally.

Autoscaling is on by default.

See also

[setScale\(\)](#), [autoScale\(\)](#)

12.29.4.40 `void QwtKnob::setBorderWidth (int bw)`

Set the knob's border width.

Parameters

<i>bw</i>	new border width
-----------	------------------

12.29.4.41 void QwtKnob::setKnobWidth (int *w*)

Change the knob's width.

The specified width must be ≥ 5 , or it will be clipped.

Parameters

<i>w</i>	New width
----------	-----------

12.29.4.42 void QwtAbstractSlider::setMass (double *val*) [virtual, inherited]

Set the slider's mass for flywheel effect.

If the slider's mass is greater than 0, it will continue to move after the mouse button has been released. Its speed decreases with time at a rate depending on the slider's mass. A large mass means that it will continue to move for a long time.

Derived widgets may overload this function to make it public.

Parameters

<i>val</i>	New mass in kg
------------	----------------

See also

[mass\(\)](#)

Reimplemented in [QwtWheel](#).

12.29.4.43 void QwtAbstractSlider::setOrientation (Qt::Orientation *o*) [virtual, inherited]

Set the orientation.

Parameters

<i>o</i>	Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.
----------	--

Reimplemented in [QwtSlider](#), and [QwtWheel](#).

12.29.4.44 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.29.4.45 void QwtAbstractSlider::setPosition (const QPoint & p)
[protected, virtual, inherited]

Move the slider to a specified point, adjust the value and emit signals if necessary.

12.29.4.46 void QwtDoubleRange::setRange (double vmin, double vmax,
double vstep = 0.0, int pageSize = 1) [inherited]

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will *not* be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.29.4.47 void QwtAbstractSlider::setReadOnly (bool readOnly)
[virtual, slot, inherited]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also

[isReadOnly\(\)](#)

12.29.4.48 void QwtAbstractScale::setScale (const QwtScaleDiv & *scaleDiv*) [**inherited**]

Specify a scale.

Disable autoscaling and define a scale by a scale division

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

See also

[setAutoScale\(\)](#)

12.29.4.49 void QwtAbstractScale::setScale (double *vmin*, double *vmax*, double *stepSize* = 0.0) [**inherited**]

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>vmin</i>	lower limit of the scale interval
<i>vmax</i>	upper limit of the scale interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.29.4.50 void QwtAbstractScale::setScale (const QwtDoubleInterval & *interval*, double *stepSize* = 0.0) [**inherited**]

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>interval</i>	Interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.29.4.51 void QwtKnob::setScaleDraw (QwtRoundScaleDraw * *scaleDraw*)

Change the scale draw of the knob

For changing the labels of the scales, it is necessary to derive from [QwtRoundScaleDraw](#) and overload [QwtRoundScaleDraw::label\(\)](#).

See also

[scaleDraw\(\)](#)

12.29.4.52 void QwtAbstractScale::setScaleEngine (QwtScaleEngine * *scaleEngine*) [inherited]

Set a scale engine.

The scale engine is responsible for calculating the scale division, and in case of auto scaling how to align the scale.

scaleEngine has to be created with new and will be deleted in `~QwtAbstractScale` or the next call of `setScaleEngine`.

12.29.4.53 void QwtAbstractScale::setScaleMaxMajor (int *ticks*) [inherited]

Set the maximum number of major tick intervals.

The scale's major ticks are calculated automatically such that the number of major intervals does not exceed *ticks*. The default value is 5.

Parameters

<i>ticks</i>	maximal number of major ticks.
--------------	--------------------------------

See also

[QwtAbstractScaleDraw](#)

**12.29.4.54 void QwtAbstractScale::setScaleMaxMinor (int *ticks*)
[*inherited*]**

Set the maximum number of minor tick intervals.

The scale's minor ticks are calculated automatically such that the number of minor intervals does not exceed ticks. The default value is 3.

Parameters

<i>ticks</i>

See also

[QwtAbstractScaleDraw](#)

12.29.4.55 void QwtDoubleRange::setStep (double *vstep*) [*inherited*]

Change the step raster.

Parameters

<i>vstep</i> new step width

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

12.29.4.56 void QwtKnob::setSymbol (QwtKnob::Symbol *s*)

Set the symbol of the knob.

See also

[symbol\(\)](#)

12.29.4.57 void QwtKnob::setTotalAngle (double *angle*)

Set the total angle by which the knob can be turned.

Parameters

<i>angle</i>	Angle in degrees.
--------------	-------------------

The default angle is 270 degrees. It is possible to specify an angle of more than 360 degrees so that the knob can be turned several times around its axis.

12.29.4.58 void QwtAbstractSlider::setTracking (bool *enable*) [*inherited*]

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i>	true (enable) or false (disable) tracking.
---------------	--

12.29.4.59 void QwtAbstractSlider::setUpdateTime (int *t*) [*inherited*]

Specify the update interval for automatic scrolling.

Parameters

<i>t</i>	update interval in milliseconds
----------	---------------------------------

See also

[getScrollMode\(\)](#)

12.29.4.60 void QwtAbstractSlider::setValid (bool *valid*) [*inline*, *inherited*]

Parameters

<i>valid</i>	true/false
--------------	------------

See also

[QwtDbfRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.29.4.61 `void QwtAbstractSlider::setValue (double val) [virtual, slot, inherited]`

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i> new value

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.29.4.62 `QSize QwtKnob::sizeHint () const [virtual]`

Returns

[minimumSizeHint\(\)](#)

12.29.4.63 `void QwtAbstractSlider::sliderMoved (double value) [signal, inherited]`

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i> new value

12.29.4.64 `void QwtAbstractSlider::sliderPressed () [signal, inherited]`

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.29.4.65 void QwtAbstractSlider::sliderReleased () [signal, inherited]

This signal is emitted when the user releases the movable part of the slider.

12.29.4.66 double QwtDoubleRange::step () const [inherited]

Returns

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.29.4.67 void QwtDoubleRange::stepChange () [protected, virtual, inherited]

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.29.4.68 void QwtAbstractSlider::stopMoving () [inherited]

Stop updating if automatic scrolling is active.

12.29.4.69 QwtKnob::Symbol QwtKnob::symbol () const

Returns

symbol of the knob

See also

[setSymbol\(\)](#)

12.29.4.70 void QwtAbstractSlider::timerEvent (QTimerEvent * *e*)
[protected, virtual, inherited]

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.29.4.71 double QwtKnob::totalAngle () const

Return the total angle.

12.29.4.72 double QwtDoubleRange::value () const [inherited]

Returns the current value.

Reimplemented in [QwtCounter](#).

12.29.4.73 void QwtAbstractSlider::valueChanged (double *value*)
[signal, inherited]

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.29.4.74 void QwtAbstractSlider::wheelEvent (QWheelEvent * *e*)
[protected, virtual, inherited]

Wheel Event handler

Parameters

<i>e</i>	Wheel event
----------	-------------

12.30 QwtLegend Class Reference

The legend widget.

```
#include <qwt_legend.h>
```

Public Types

- enum [LegendDisplayPolicy](#) {
 NoIdentifier = 0,
 FixedIdentifier = 1,
 AutoIdentifier = 2 }
- enum [LegendItemMode](#) {
 ReadOnlyItem,
 ClickableItem,
 CheckableItem }

Public Member Functions

- void [clear](#) ()
- const QWidget * [contentsWidget](#) () const
- QWidget * [contentsWidget](#) ()
- [LegendDisplayPolicy](#) [displayPolicy](#) () const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- [QwtLegendItemManager](#) * [find](#) (const QWidget *) const
- QWidget * [find](#) (const [QwtLegendItemManager](#) *) const
- virtual int [heightForWidth](#) (int w) const
- QScrollBar * [horizontalScrollBar](#) () const
- int [identifierMode](#) () const
- void [insert](#) (const [QwtLegendItemManager](#) *, QWidget *)
- bool [isEmpty](#) () const
- uint [itemCount](#) () const
- [LegendItemMode](#) [itemMode](#) () const
- virtual QList< QWidget * > [legendItems](#) () const
- [QwtLegend](#) (QWidget *parent=NULL)
- void [remove](#) (const [QwtLegendItemManager](#) *)
- void [setDisplayPolicy](#) ([LegendDisplayPolicy](#) policy, int mode)
- void [setItemMode](#) ([LegendItemMode](#))
- virtual QSize [sizeHint](#) () const
- QScrollBar * [verticalScrollBar](#) () const
- virtual [~QwtLegend](#) ()

Protected Member Functions

- virtual void [layoutContents](#) ()
- virtual void [resizeEvent](#) (QResizeEvent *)

12.30.1 Detailed Description

The legend widget. The [QwtLegend](#) widget is a tabular arrangement of legend items. Legend items might be any type of widget, but in general they will be a [QwtLegendItem](#).

See also

[QwtLegendItem](#), [QwtLegendItemManager](#) [QwtPlot](#)

12.30.2 Member Enumeration Documentation

12.30.2.1 enum QwtLegend::LegendDisplayPolicy

Display policy.

- NoIdentifier
The client code is responsible how to display of each legend item. The Qwt library will not interfere.
- FixedIdentifier
All legend items are displayed with the [QwtLegendItem::IdentifierMode](#) to be passed in 'mode'.
- AutoIdentifier
Each legend item is displayed with a mode that is a bitwise or of
 - [QwtLegendItem::ShowLine](#) (if its curve is drawn with a line) and
 - [QwtLegendItem::ShowSymbol](#) (if its curve is drawn with symbols) and
 - [QwtLegendItem::ShowText](#) (if the has a title).

Default is AutoIdentifier.

See also

[setDisplayPolicy\(\)](#), [displayPolicy\(\)](#), [QwtLegendItem::IdentifierMode](#)

12.30.2.2 enum QwtLegend::LegendItemMode

Interaction mode for the legend items.

- ReadOnlyItem
The legend item is not interactive, like a label

- ClickableItem

The legend item is clickable, like a push button

- CheckableItem

The legend item is checkable, like a checkable button

Default is ReadOnlyItem.

See also

[setItemMode\(\)](#), [itemMode\(\)](#), [QwtLegendItem::IdentifierMode](#) [QwtLegendItem::clicked\(\)](#),
[QwtLegendItem::checked\(\)](#), [QwtPlot::legendClicked\(\)](#), [QwtPlot::legendChecked\(\)](#)

12.30.3 Constructor & Destructor Documentation

12.30.3.1 QwtLegend::QwtLegend (QWidget * *parent* = *NULL*) [explicit]

Constructor

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.30.3.2 QwtLegend::~~QwtLegend () [virtual]

Destructor.

12.30.4 Member Function Documentation

12.30.4.1 void QwtLegend::clear ()

Remove all items.

12.30.4.2 QWidget * QwtLegend::contentsWidget ()

The contents widget is the only child of the viewport() and the parent widget of all legend items.

12.30.4.3 `const QWidget * QwtLegend::contentsWidget () const`

The contents widget is the only child of the viewport() and the parent widget of all legend items.

12.30.4.4 `QwtLegend::LegendDisplayPolicy QwtLegend::displayPolicy () const`**Returns**

the legend display policy. Default is LegendDisplayPolicy::Auto.

See also

[setDisplayPolicy\(\)](#), [LegendDisplayPolicy](#)

12.30.4.5 `bool QwtLegend::eventFilter (QObject * o, QEvent * e) [virtual]`

Filter layout related events of [QwtLegend::contentsWidget\(\)](#).

Parameters

<i>o</i>	Object to be filtered
<i>e</i>	Event

12.30.4.6 `QWidget * QwtLegend::find (const QwtLegendItemManager * plotItem) const`

Find the widget that represents a plot item

Parameters

<i>plotItem</i>	Plot item
-----------------	-----------

Returns

Widget on the legend, or NULL

12.30.4.7 `QwtLegendItemManager * QwtLegend::find (const QWidget * legendItem) const`

Find the widget that represents a plot item

Parameters

<i>legendItem</i>	Legend item
-------------------	-------------

Returns

Widget on the legend, or NULL

12.30.4.8 int QwtLegend::heightForWidth (int *width*) const [virtual]**Returns**

The preferred height, for the width w.

Parameters

<i>width</i>	Width
--------------	-------

12.30.4.9 QScrollBar * QwtLegend::horizontalScrollBar () const**Returns**

Horizontal scrollbar

See also

[verticalScrollBar\(\)](#)

12.30.4.10 int QwtLegend::identifierMode () const**Returns**

the IdentifierMode to be used in combination with LegendDisplayPolicy::Fixed.

Default is ShowLine | ShowSymbol | ShowText.

12.30.4.11 void QwtLegend::insert (const QwtLegendItemManager * *plotItem*, QWidget * *legendItem*)

Insert a new item for a plot item

Parameters

<i>plotItem</i>	Plot item
<i>legendItem</i>	New legend item

Note

The parent of item will be changed to [QwtLegend::contentsWidget\(\)](#)

12.30.4.12 bool QwtLegend::isEmpty () const

Return true, if there are no legend items.

12.30.4.13 uint QwtLegend::itemCount () const

Return the number of legend items.

12.30.4.14 QwtLegend::LegendItemMode QwtLegend::itemMode () const**See also**

[LegendItemMode](#)

12.30.4.15 void QwtLegend::layoutContents () [protected, virtual]

Adjust contents widget and item layout to the size of the viewport().

**12.30.4.16 QList< QWidget * > QwtLegend::legendItems () const
[virtual]**

Return a list of all legend items.

**12.30.4.17 void QwtLegend::remove (const QwtLegendItemManager *
plotItem)**

Find the corresponding item for a *plotItem* and remove it from the item list.

Parameters

<i>plotItem</i> Plot item

12.30.4.18 `void QwtLegend::resizeEvent (QResizeEvent * e)` [**protected**, **virtual**]

Resize event

Parameters

<i>e</i>	Resize event
----------	--------------

12.30.4.19 `void QwtLegend::setDisplayPolicy (LegendDisplayPolicy policy, int mode)`

Set the legend display policy to:

Parameters

<i>policy</i>	Legend display policy
<i>mode</i>	Identifier mode (or'd ShowLine, ShowSymbol, ShowText)

See also

[displayPolicy\(\)](#), [LegendDisplayPolicy](#)

12.30.4.20 `void QwtLegend::setItemMode (LegendItemMode mode)`

See also

[LegendItemMode](#)

12.30.4.21 `QSize QwtLegend::sizeHint () const` [**virtual**]

Return a size hint.

12.30.4.22 `QScrollBar * QwtLegend::verticalScrollBar () const`

Returns

Vertical scrollbar

See also

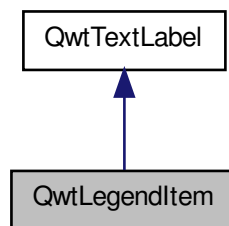
[horizontalScrollBar\(\)](#)

12.31 QwtLegendItem Class Reference

A legend label.

```
#include <qwt_legend_item.h>
```

Inheritance diagram for QwtLegendItem:



Public Types

- enum `IdentifierMode` {
 NoIdentifier = 0,
 ShowLine = 1,
 ShowSymbol = 2,
 ShowText = 4 }

Public Slots

- void `clear` ()
- void `setChecked` (bool on)
- void `setText` (const QString &, `QwtText::TextFormat` textFormat=`QwtText::AutoText`)

Signals

- void `checked` (bool)
- void `clicked` ()
- void `pressed` ()
- void `released` ()

Public Member Functions

- const QPen & [curvePen](#) () const
- virtual void [drawIdentifier](#) (QPainter *, const QRect &) const
- virtual void [drawItem](#) (QPainter *p, const QRect &) const
- virtual int [heightForWidth](#) (int) const
- int [identifierMode](#) () const
- int [identifierWidth](#) () const
- int [indent](#) () const
- bool [isChecked](#) () const
- [QwtLegend::LegendItemMode](#) [itemMode](#) () const
- int [margin](#) () const
- virtual QSize [minimumSizeHint](#) () const
- [QwtLegendItem](#) (QWidget *parent=0)
- [QwtLegendItem](#) (const [QwtSymbol](#) &, const QPen &, const [QwtText](#) &, QWidget *parent=0)
- void [setCurvePen](#) (const QPen &)
- void [setIdentifierMode](#) (int)
- void [setIdentifierWidth](#) (int width)
- void [setIndent](#) (int)
- void [setItemMode](#) ([QwtLegend::LegendItemMode](#))
- void [setMargin](#) (int)
- void [setSpacing](#) (int spacing)
- void [setSymbol](#) (const [QwtSymbol](#) &)
- virtual void [setText](#) (const [QwtText](#) &)
- virtual QSize [sizeHint](#) () const
- int [spacing](#) () const
- const [QwtSymbol](#) & [symbol](#) () const
- const [QwtText](#) & [text](#) () const
- QRect [textRect](#) () const
- virtual ~[QwtLegendItem](#) ()

Protected Member Functions

- virtual void [drawContents](#) (QPainter *)
- virtual void [drawText](#) (QPainter *, const QRect &)
- bool [isDown](#) () const
- virtual void [keyPressEvent](#) (QKeyEvent *)
- virtual void [keyReleaseEvent](#) (QKeyEvent *)
- virtual void [mousePressEvent](#) (QMouseEvent *)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *)
- virtual void [paintEvent](#) (QPaintEvent *)
- void [setDown](#) (bool)

12.31.1 Detailed Description

A legend label. [QwtLegendItem](#) represents a curve on a legend. It displays an curve identifier with an explaining text. The identifier might be a combination of curve symbol and line. In readonly mode it behaves like a label, otherwise like an unstylish push button.

See also

[QwtLegend](#), [QwtPlotCurve](#)

12.31.2 Member Enumeration Documentation

12.31.2.1 enum QwtLegendItem::IdentifierMode

Identifier mode.

Default is ShowLine | ShowText

See also

[identifierMode\(\)](#), [setIdentifierMode\(\)](#)

12.31.3 Constructor & Destructor Documentation

12.31.3.1 QwtLegendItem::QwtLegendItem (QWidget * *parent* = 0)
[explicit]

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.31.3.2 QwtLegendItem::QwtLegendItem (const QwtSymbol & *symbol*,
const QPen & *curvePen*, const QwtText & *text*, QWidget * *parent* = 0)
[explicit]

Parameters

<i>symbol</i>	Curve symbol
<i>curvePen</i>	Curve pen
<i>text</i>	Label text
<i>parent</i>	Parent widget

12.31.3.3 QwtLegendItem::~~QwtLegendItem () [virtual]

Destructor.

12.31.4 Member Function Documentation**12.31.4.1 void QwtLegendItem::checked (bool) [signal]**

Signal, when the legend item has been toggled.

12.31.4.2 void QwtTextLabel::clear () [slot, inherited]

Clear the text and all [QwtText](#) attributes.

12.31.4.3 void QwtLegendItem::clicked () [signal]

Signal, when the legend item has been clicked.

12.31.4.4 const QPen & QwtLegendItem::curvePen () const**Returns**

The curve pen.

See also

[setCurvePen\(\)](#)

**12.31.4.5 void QwtTextLabel::drawContents (QPainter * *painter*)
[protected, virtual, inherited]**

Redraw the text and focus indicator.

**12.31.4.6 void QwtLegendItem::drawIdentifier (QPainter * *painter*, const
QRect & *rect*) const [virtual]**

Paint the identifier to a given rect.

Parameters

<i>painter</i>	Painter
<i>rect</i>	Rect where to paint

12.31.4.7 void QwtLegendItem::drawItem (QPainter * *painter*, const QRect & *rect*) const [virtual]

Draw the legend item to a given rect.

Parameters

<i>painter</i>	Painter
<i>rect</i>	Rect where to paint the button

12.31.4.8 void QwtLegendItem::drawText (QPainter * *painter*, const QRect & *textRect*) [protected, virtual]

Redraw the text.

Reimplemented from [QwtTextLabel](#).

12.31.4.9 int QwtTextLabel::heightForWidth (int *width*) const [virtual, inherited]

Returns the preferred height for this widget, given the width.

Parameters

<i>width</i>	Width
--------------	-------

12.31.4.10 int QwtLegendItem::identifierMode () const

Or'd values of IdentifierMode.

See also

[setIdentifierMode\(\)](#), [IdentifierMode](#)

12.31.4.11 int QwtLegendItem::identifierWidth () const

Return the width of the identifier

See also

[setIdentifierWidth\(\)](#)

12.31.4.12 int QwtTextLabel::indent () const [inherited]

Return label's text indent in pixels.

12.31.4.13 bool QwtLegendItem::isChecked () const

Return true, if the item is checked.

12.31.4.14 bool QwtLegendItem::isDown () const [protected]

Return true, if the item is down.

12.31.4.15 QwtLegend::LegendItemMode QwtLegendItem::itemMode () const

Return the item mode

See also

[setItemMode\(\)](#)

12.31.4.16 void QwtLegendItem::keyPressEvent (QKeyEvent * e) [protected, virtual]

Handle key press events.

12.31.4.17 void QwtLegendItem::keyReleaseEvent (QKeyEvent * e) [protected, virtual]

Handle key release events.

12.31.4.18 int QwtTextLabel::margin () const [inherited]

Return label's text indent in pixels.

12.31.4.19 `QSize QwtTextLabel::minimumSizeHint () const` **[virtual, inherited]**

Return a minimum size hint.

12.31.4.20 `void QwtLegendItem::mousePressEvent (QMouseEvent * e)` **[protected, virtual]**

Handle mouse press events.

12.31.4.21 `void QwtLegendItem::mouseReleaseEvent (QMouseEvent * e)` **[protected, virtual]**

Handle mouse release events.

12.31.4.22 `void QwtLegendItem::paintEvent (QPaintEvent * e)` **[protected, virtual]**

Paint event.

Reimplemented from [QwtTextLabel](#).

12.31.4.23 `void QwtLegendItem::pressed ()` **[signal]**

Signal, when the legend item has been pressed.

12.31.4.24 `void QwtLegendItem::released ()` **[signal]**

Signal, when the legend item has been released.

12.31.4.25 `void QwtLegendItem::setChecked (bool on)` **[slot]**

Check/Uncheck a the item

Parameters

<i>on</i>	check/uncheck
-----------	---------------

See also

[setItemMode\(\)](#)

12.31.4.26 void QwtLegendItem::setCurvePen (const QPen & *pen*)

Set curve pen.

Parameters

<i>pen</i>	Curve pen
------------	-----------

See also

[curvePen\(\)](#)

12.31.4.27 void QwtLegendItem::setDown (bool *down*) [protected]

Set the item being down.

12.31.4.28 void QwtLegendItem::setIdentifierMode (int *mode*)

Set identifier mode. Default is ShowLine | ShowText.

Parameters

<i>mode</i>	Or'd values of IdentifierMode
-------------	-------------------------------

See also

[identifierMode\(\)](#)

12.31.4.29 void QwtLegendItem::setIdentifierWidth (int *width*)

Set the width for the identifier Default is 8 pixels

Parameters

<i>width</i>	New width
--------------	-----------

See also

[identifierMode\(\)](#), [identifierWidth\(\)](#)

12.31.4.30 void QwtTextLabel::setIndent (int *indent*) [inherited]

Set label's text indent in pixels

Parameters

<i>indent</i>	Indentation in pixels
---------------	-----------------------

12.31.4.31 void QwtLegendItem::setItemMode (QwtLegend::LegendItemMode *mode*)

Set the item mode The default is QwtLegend::ReadOnlyItem

Parameters

<i>mode</i>	Item mode
-------------	-----------

See also

[itemMode\(\)](#)

12.31.4.32 void QwtTextLabel::setMargin (int *margin*) [inherited]

Set label's margin in pixels

Parameters

<i>margin</i>	Margin in pixels
---------------	------------------

12.31.4.33 void QwtLegendItem::setSpacing (int *spacing*)

Change the spacing

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

[spacing\(\)](#), [identifierWidth\(\)](#), [QwtTextLabel::margin\(\)](#)

12.31.4.34 void QwtLegendItem::setSymbol (const QwtSymbol & *symbol*)

Set curve symbol.

Parameters

<i>symbol</i>	Symbol
---------------	--------

See also

[symbol\(\)](#)

12.31.4.35 `void QwtLegendItem::setText (const QwtText & text)`
[virtual]

Set the text to the legend item

Parameters

<i>text</i>	Text label
-------------	------------

See also

[QwtTextLabel::text\(\)](#)

Reimplemented from [QwtTextLabel](#).

12.31.4.36 `void QwtTextLabel::setText (const QString & text,
QwtText::TextFormat textFormat = QwtText::AutoText)`
[slot, inherited]

Change the label's text, keeping all other [QwtText](#) attributes

Parameters

<i>text</i>	New text
<i>textFormat</i>	Format of text

See also

[QwtText](#)

12.31.4.37 `QSize QwtLegendItem::sizeHint () const` **[virtual]**

Return a size hint.

Reimplemented from [QwtTextLabel](#).

12.31.4.38 `int QwtLegendItem::spacing () const`

Return the spacing

See also

[setSpacing\(\)](#), [identifierWidth\(\)](#), [QwtTextLabel::margin\(\)](#)

12.31.4.39 `const QwtSymbol & QwtLegendItem::symbol () const`**Returns**

The curve symbol.

See also

[setSymbol\(\)](#)

12.31.4.40 `const QwtText & QwtTextLabel::text () const` **[inherited]**

Return the text.

12.31.4.41 `QRect QwtTextLabel::textRect () const` **[inherited]**

Calculate the rect for the text in widget coordinates

Returns

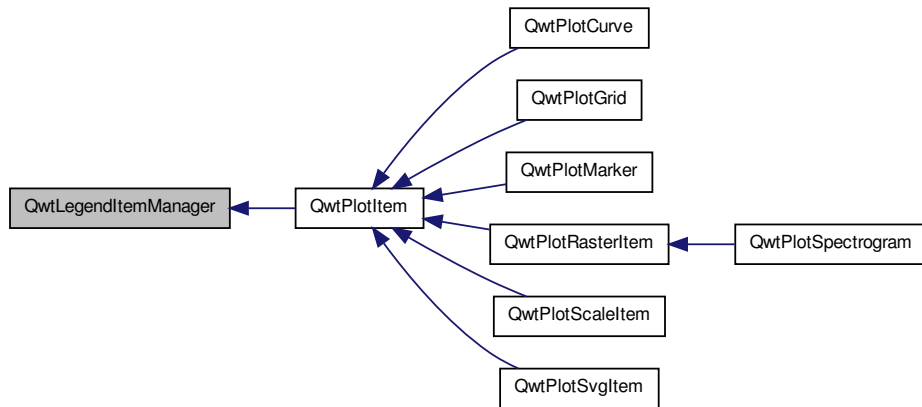
Text rect

12.32 QwtLegendItemManager Class Reference

Abstract API to bind plot items to the legend.

```
#include <qwt_legend_itemmanager.h>
```

Inheritance diagram for QwtLegendItemManager:



Public Member Functions

- virtual QWidget * [legendItem](#) () const =0
- [QwtLegendItemManager](#) ()
- virtual void [updateLegend](#) (QwtLegend *legend) const =0
- virtual [~QwtLegendItemManager](#) ()

12.32.1 Detailed Description

Abstract API to bind plot items to the legend.

12.32.2 Constructor & Destructor Documentation

12.32.2.1 QwtLegendItemManager::QwtLegendItemManager () [inline]

Constructor.

12.32.2.2 virtual QwtLegendItemManager::~~QwtLegendItemManager () [inline, virtual]

Destructor.

12.32.3 Member Function Documentation

12.32.3.1 `virtual QWidget* QwtLegendItemManager::legendItem () const` `[pure virtual]`

Allocate the widget that represents the item on the legend

Returns

Allocated widget

See also

[updateLegend\(\)](#) `QwtLegend()`

Implemented in [QwtPlotItem](#).

12.32.3.2 `virtual void QwtLegendItemManager::updateLegend (QwtLegend * legend) const` `[pure virtual]`

Update the widget that represents the item on the legend

Parameters

<i>legend</i> Legend

See also

[legendItem\(\)](#)

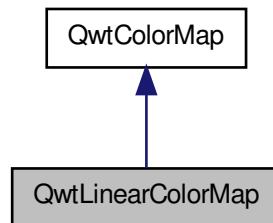
Implemented in [QwtPlotCurve](#), and [QwtPlotItem](#).

12.33 QwtLinearColorMap Class Reference

[QwtLinearColorMap](#) builds a color map from color stops.

```
#include <qwt_color_map.h>
```

Inheritance diagram for QwtLinearColorMap:



Public Types

- enum [Format](#) {
 RGB,
 Indexed }
- enum [Mode](#) {
 FixedColors,
 ScaledColors }

Public Member Functions

- void [addColorStop](#) (double value, const QColor &)
- QColor [color](#) (const [QwtDoubleInterval](#) &, double value) const
- QColor [color1](#) () const
- QColor [color2](#) () const
- virtual unsigned char [colorIndex](#) (const [QwtDoubleInterval](#) &, double value) const
- QwtArray< double > [colorStops](#) () const
- virtual QVector< QRgb > [colorTable](#) (const [QwtDoubleInterval](#) &) const
- virtual [QwtColorMap](#) * [copy](#) () const
- [Format](#) [format](#) () const
- [Mode](#) [mode](#) () const
- [QwtLinearColorMap](#) & [operator=](#) (const [QwtLinearColorMap](#) &)
- [QwtLinearColorMap](#) (const QColor &from, const QColor &to, [QwtColorMap::Format](#)=[QwtColorMap::RGB](#))
- [QwtLinearColorMap](#) ([QwtColorMap::Format](#)=[QwtColorMap::RGB](#))
- [QwtLinearColorMap](#) (const [QwtLinearColorMap](#) &)
- virtual QRgb [rgb](#) (const [QwtDoubleInterval](#) &, double value) const
- void [setColorInterval](#) (const QColor &color1, const QColor &color2)
- void [setMode](#) ([Mode](#))
- virtual [~QwtLinearColorMap](#) ()

12.33.1 Detailed Description

[QwtLinearColorMap](#) builds a color map from color stops. A color stop is a color at a specific position. The valid range for the positions is [0.0, 1.0]. When mapping a value into a color it is translated into this interval. If `mode() == FixedColors` the color is calculated from the next lower color stop. If `mode() == ScaledColors` the color is calculated by interpolating the colors of the adjacent stops.

12.33.2 Member Enumeration Documentation

12.33.2.1 enum QwtColorMap::Format **[inherited]**

- RGB

The map is intended to map into QRgb values.

- Indexed

The map is intended to map into 8 bit values, that are indices into the color table.

See also

[rgb\(\)](#), [colorIndex\(\)](#), [colorTable\(\)](#)

12.33.2.2 enum QwtLinearColorMap::Mode

Mode of color map

See also

[setMode\(\)](#), [mode\(\)](#)

12.33.3 Constructor & Destructor Documentation

12.33.3.1 QwtLinearColorMap::QwtLinearColorMap (
 QwtColorMap::Format *format* = *QwtColorMap::RGB*)

Build a color map with two stops at 0.0 and 1.0. The color at 0.0 is Qt::blue, at 1.0 it is Qt::yellow.

Parameters

<i>format</i>	Preferred format of the color map
---------------	-----------------------------------

12.33.3.2 QwtLinearColorMap::QwtLinearColorMap (const QColor & *color1*, const QColor & *color2*, QwtColorMap::Format *format* = *QwtColorMap::RGB*)

Build a color map with two stops at 0.0 and 1.0.

Parameters

<i>color1</i>	Color used for the minimum value of the value interval
<i>color2</i>	Color used for the maximum value of the value interval
<i>format</i>	Preferred format of the color map

12.33.3.3 QwtLinearColorMap::QwtLinearColorMap (const QwtLinearColorMap & *other*)

Copy constructor.

12.33.3.4 QwtLinearColorMap::~QwtLinearColorMap () [virtual]

Destructor.

12.33.4 Member Function Documentation

12.33.4.1 void QwtLinearColorMap::addColorStop (double *value*, const QColor & *color*)

Add a color stop

The value has to be in the range [0.0, 1.0]. F.e. a stop at position 17.0 for a range [10.0,20.0] must be passed as: (17.0 - 10.0) / (20.0 - 10.0)

Parameters

<i>value</i>	Value between [0.0, 1.0]
<i>color</i>	Color stop

12.33.4.2 QColor QwtColorMap::color (const QwtDoubleInterval & *interval*, double *value*) const [inline, inherited]

Map a value into a color

Parameters

<i>interval</i>	Valid interval for values
<i>value</i>	Value

Returns

Color corresponding to value

Warning

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

12.33.4.3 QColor QwtLinearColorMap::color1 () const**Returns**

the first color of the color range

See also

[setColorInterval\(\)](#)

12.33.4.4 QColor QwtLinearColorMap::color2 () const**Returns**

the second color of the color range

See also

[setColorInterval\(\)](#)

12.33.4.5 unsigned char QwtLinearColorMap::colorIndex (const QwtDoubleInterval & *interval*, double *value*) const [virtual]

Map a value of a given interval into a color index, between 0 and 255

Parameters

<i>interval</i>	Range for all values
<i>value</i>	Value to map into a color index

Implements [QwtColorMap](#).

12.33.4.6 QwtArray< double > QwtLinearColorMap::colorStops () const

Return all positions of color stops in increasing order

12.33.4.7 QwtColorTable QwtColorMap::colorTable (const QwtDoubleInterval & *interval*) const [virtual, inherited]

Build and return a color map of 256 colors

The color table is needed for rendering indexed images in combination with using [colorIndex\(\)](#).

Parameters

<i>interval</i>	Range for the values
-----------------	----------------------

Returns

A color table, that can be used for a QImage

12.33.4.8 QwtColorMap * QwtLinearColorMap::copy () const [virtual]

Clone the color map.

Implements [QwtColorMap](#).

12.33.4.9 QwtColorMap::Format QwtColorMap::format () const [inline, inherited]

Returns

Intended format of the color map

See also

[Format](#)

12.33.4.10 QwtLinearColorMap::Mode QwtLinearColorMap::mode () const

Returns

Mode of the color map

See also

[setMode\(\)](#)

12.33.4.11 QwtLinearColorMap & QwtLinearColorMap::operator= (const QwtLinearColorMap & *other*)

Assignment operator.

12.33.4.12 QRgb QwtLinearColorMap::rgb (const QwtDoubleInterval & *interval*, double *value*) const [virtual]

Map a value of a given interval into a rgb value

Parameters

<i>interval</i>	Range for all values
<i>value</i>	Value to map into a rgb value

Implements [QwtColorMap](#).

12.33.4.13 void QwtLinearColorMap::setColorInterval (const QColor & *color1*, const QColor & *color2*)

Set the color range

Add stops at 0.0 and 1.0.

Parameters

<i>color1</i>	Color used for the minimum value of the value interval
<i>color2</i>	Color used for the maximum value of the value interval

See also

[color1\(\)](#), [color2\(\)](#)

12.33.4.14 void QwtLinearColorMap::setMode (Mode *mode*)

Set the mode of the color map.

FixedColors means the color is calculated from the next lower color stop. ScaledColors means the color is calculated by interpolating the colors of the adjacent stops.

See also

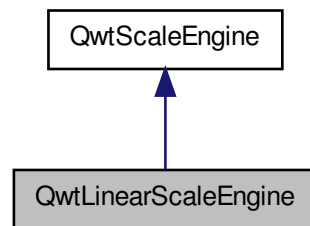
[mode\(\)](#)

12.34 QwtLinearScaleEngine Class Reference

A scale engine for linear scales.

```
#include <qwt_scale_engine.h>
```

Inheritance diagram for QwtLinearScaleEngine:



Public Types

- enum [Attribute](#) {
 NoAttribute = 0,
 IncludeReference = 1,
 Symmetric = 2,
 Floating = 4,
 Inverted = 8 }

Public Member Functions

- int [attributes](#) () const
- virtual void [autoScale](#) (int maxSteps, double &x1, double &x2, double &stepSize) const
- virtual [QwtScaleDiv](#) [divideScale](#) (double x1, double x2, int numMajorSteps, int numMinorSteps, double stepSize=0.0) const
- double [lowerMargin](#) () const
- double [reference](#) () const
- void [setAttribute](#) ([Attribute](#), bool on=true)
- void [setAttributes](#) (int)
- void [setMargins](#) (double lower, double upper)
- void [setReference](#) (double reference)
- bool [testAttribute](#) ([Attribute](#)) const
- virtual [QwtScaleTransformation](#) * [transformation](#) () const
- double [upperMargin](#) () const

Protected Member Functions

- [QwtDoubleInterval align](#) (const [QwtDoubleInterval](#) &, double stepSize) const
- [QwtDoubleInterval buildInterval](#) (double v) const
- bool [contains](#) (const [QwtDoubleInterval](#) &, double val) const
- double [divideInterval](#) (double interval, int numSteps) const
- [QwtValueList strip](#) (const [QwtValueList](#) &, const [QwtDoubleInterval](#) &) const

12.34.1 Detailed Description

A scale engine for linear scales. The step size will fit into the pattern $\{1, 2, 5\} \cdot 10^n$, where n is an integer.

12.34.2 Member Enumeration Documentation**12.34.2.1 enum QwtScaleEngine::Attribute [inherited]**

- IncludeReference
Build a scale which includes the [reference\(\)](#) value.
- Symmetric
Build a scale which is symmetric to the [reference\(\)](#) value.
- Floating
The endpoints of the scale are supposed to be equal the outmost included values plus the specified margins (see [setMargins\(\)](#)). If this attribute is *not* set, the endpoints of the scale will be integer multiples of the step size.
- Inverted
Turn the scale upside down.

See also

[setAttribute\(\)](#), [testAttribute\(\)](#), [reference\(\)](#), [lowerMargin\(\)](#), [upperMargin\(\)](#)

12.34.3 Member Function Documentation**12.34.3.1 QwtDoubleInterval QwtLinearScaleEngine::align (const QwtDoubleInterval & *interval*, double *stepSize*) const [protected]**

Align an interval to a step size.

The limits of an interval are aligned that both are integer multiples of the step size.

Parameters

<i>interval</i>	Interval
<i>stepSize</i>	Step size

Returns

Aligned interval

12.34.3.2 int QwtScaleEngine::attributes () const [inherited]

Return the scale attributes

See also

[Attribute](#), [setAttributes\(\)](#), [testAttribute\(\)](#)

12.34.3.3 void QwtLinearScaleEngine::autoScale (int *maxNumSteps*, double & *x1*, double & *x2*, double & *stepSize*) const [virtual]

Align and divide an interval

Parameters

<i>maxNumSteps</i>	Max. number of steps
<i>x1</i>	First limit of the interval (In/Out)
<i>x2</i>	Second limit of the interval (In/Out)
<i>stepSize</i>	Step size (Out)

See also

[setAttribute\(\)](#)

Implements [QwtScaleEngine](#).

12.34.3.4 QwtDoubleInterval QwtScaleEngine::buildInterval (double *v*) const [protected, inherited]

Build an interval for a value.

In case of $v == 0.0$ the interval is $[-0.5, 0.5]$, otherwise it is $[0.5 * v, 1.5 * v]$

12.34.3.5 bool QwtScaleEngine::contains (const QwtDoubleInterval & *interval*, double *value*) const [protected, inherited]

Check if an interval "contains" a value

Parameters

<i>interval</i>	Interval
<i>value</i>	Value

See also

[QwtScaleArithmetic::compareEps\(\)](#)

12.34.3.6 double QwtScaleEngine::divideInterval (double *intervalSize*, int *numSteps*) const **[protected, inherited]**

Calculate a step size for an interval size

Parameters

<i>intervalSize</i>	Interval size
<i>numSteps</i>	Number of steps

Returns

Step size

12.34.3.7 QwtScaleDiv QwtLinearScaleEngine::divideScale (double *x1*, double *x2*, int *maxMajSteps*, int *maxMinSteps*, double *stepSize* = 0.0) const **[virtual]**

Calculate a scale division.

Parameters

<i>x1</i>	First interval limit
<i>x2</i>	Second interval limit
<i>maxMajSteps</i>	Maximum for the number of major steps
<i>maxMinSteps</i>	Maximum number of minor steps
<i>stepSize</i>	Step size. If stepSize == 0, the scaleEngine calculates one.

See also

[QwtScaleEngine::stepSize\(\)](#), [QwtScaleEngine::subDivide\(\)](#)

Implements [QwtScaleEngine](#).

12.34.3.8 double QwtScaleEngine::lowerMargin () const **[inherited]**

Returns

the margin at the lower end of the scale The default margin is 0.

See also

[setMargins\(\)](#)

12.34.3.9 double QwtScaleEngine::reference () const [inherited]**Returns**

the reference value

See also

[setReference\(\)](#), [setAttribute\(\)](#)

12.34.3.10 void QwtScaleEngine::setAttribute (Attribute *attribute*, bool *on* = true) [inherited]

Change a scale attribute

Parameters

<i>attribute</i>	Attribute to change
<i>on</i>	On/Off

See also

[Attribute](#), [testAttribute\(\)](#)

12.34.3.11 void QwtScaleEngine::setAttributes (int *attributes*) [inherited]

Change the scale attribute

Parameters

<i>attributes</i>	Set scale attributes
-------------------	----------------------

See also

[Attribute](#), [attributes\(\)](#)

12.34.3.12 void QwtScaleEngine::setMargins (double *lower*, double *upper*) [*inherited*]

Specify margins at the scale's endpoints.

Parameters

<i>lower</i>	minimum distance between the scale's lower boundary and the smallest enclosed value
<i>upper</i>	minimum distance between the scale's upper boundary and the greatest enclosed value

Margins can be used to leave a minimum amount of space between the enclosed intervals and the boundaries of the scale.

Warning

- [QwtLog10ScaleEngine](#) measures the margins in decades.

See also

[upperMargin\(\)](#), [lowerMargin\(\)](#)

12.34.3.13 void QwtScaleEngine::setReference (double *r*) [*inherited*]

Specify a reference point.

Parameters

<i>r</i>	new reference value
----------	---------------------

The reference point is needed if options IncludeReference or Symmetric are active. Its default value is 0.0.

See also

[Attribute](#)

12.34.3.14 QwtValueList QwtScaleEngine::strip (const QwtValueList & *ticks*, const QwtDoubleInterval & *interval*) const [*protected*, *inherited*]

Remove ticks from a list, that are not inside an interval

Parameters

<i>ticks</i>	Tick list
<i>interval</i>	Interval

Returns

Stripped tick list

12.34.3.15 `bool QwtScaleEngine::testAttribute (Attribute attribute) const`
[*inherited*]

Check if a attribute is set.

Parameters

<i>attribute</i>	Attribute to be tested
------------------	------------------------

See also

[Attribute](#), [setAttribute\(\)](#)

12.34.3.16 `QwtScaleTransformation * QwtLinearScaleEngine::transformation`
`() const` **[*virtual*]**

Return a transformation, for linear scales

Implements [QwtScaleEngine](#).

12.34.3.17 `double QwtScaleEngine::upperMargin () const` **[*inherited*]**

Returns

the margin at the upper end of the scale The default margin is 0.

See also

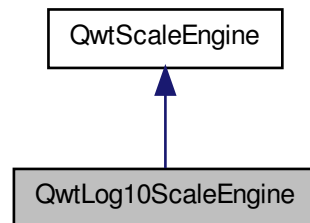
[setMargins\(\)](#)

12.35 QwtLog10ScaleEngine Class Reference

A scale engine for logarithmic (base 10) scales.

```
#include <qwt_scale_engine.h>
```

Inheritance diagram for QwtLog10ScaleEngine:



Public Types

- enum [Attribute](#) {
 NoAttribute = 0,
 IncludeReference = 1,
 Symmetric = 2,
 Floating = 4,
 Inverted = 8 }

Public Member Functions

- int [attributes](#) () const
- virtual void [autoScale](#) (int maxSteps, double &x1, double &x2, double &step-Size) const
- virtual [QwtScaleDiv](#) [divideScale](#) (double x1, double x2, int numMajorSteps, int numMinorSteps, double stepSize=0.0) const
- double [lowerMargin](#) () const
- double [reference](#) () const
- void [setAttribute](#) ([Attribute](#), bool on=true)
- void [setAttributes](#) (int)
- void [setMargins](#) (double lower, double upper)
- void [setReference](#) (double reference)
- bool [testAttribute](#) ([Attribute](#)) const
- virtual [QwtScaleTransformation](#) * [transformation](#) () const
- double [upperMargin](#) () const

Protected Member Functions

- [QwtDoubleInterval buildInterval](#) (double v) const
- bool [contains](#) (const [QwtDoubleInterval](#) &, double val) const
- double [divideInterval](#) (double interval, int numSteps) const
- [QwtDoubleInterval log10](#) (const [QwtDoubleInterval](#) &) const
- [QwtDoubleInterval pow10](#) (const [QwtDoubleInterval](#) &) const
- [QwtValueList strip](#) (const [QwtValueList](#) &, const [QwtDoubleInterval](#) &) const

12.35.1 Detailed Description

A scale engine for logarithmic (base 10) scales. The step size is measured in **decades** and the major step size will be adjusted to fit the pattern $\{1, 2, 3, 5\} \cdot 10^n$, where n is a natural number including zero.

Warning

the step size as well as the margins are measured in **decades**.

12.35.2 Member Enumeration Documentation

12.35.2.1 enum QwtScaleEngine::Attribute [inherited]

- IncludeReference
Build a scale which includes the [reference\(\)](#) value.
- Symmetric
Build a scale which is symmetric to the [reference\(\)](#) value.
- Floating
The endpoints of the scale are supposed to be equal the outmost included values plus the specified margins (see [setMargins\(\)](#)). If this attribute is **not** set, the endpoints of the scale will be integer multiples of the step size.
- Inverted
Turn the scale upside down.

See also

[setAttribute\(\)](#), [testAttribute\(\)](#), [reference\(\)](#), [lowerMargin\(\)](#), [upperMargin\(\)](#)

12.35.3 Member Function Documentation

12.35.3.1 `int QwtScaleEngine::attributes () const` `[inherited]`

Return the scale attributes

See also

[Attribute](#), [setAttributees\(\)](#), [testAttribute\(\)](#)12.35.3.2 `void QwtLog10ScaleEngine::autoScale (int maxNumSteps, double & x1, double & x2, double & stepSize) const` `[virtual]`

Align and divide an interval

Parameters

<i>maxNumSteps</i>	Max. number of steps
<i>x1</i>	First limit of the interval (In/Out)
<i>x2</i>	Second limit of the interval (In/Out)
<i>stepSize</i>	Step size (Out)

See also

[QwtScaleEngine::setAttribute\(\)](#)Implements [QwtScaleEngine](#).12.35.3.3 `QwtDoubleInterval QwtScaleEngine::buildInterval (double v) const` `[protected, inherited]`

Build an interval for a value.

In case of $v == 0.0$ the interval is $[-0.5, 0.5]$, otherwise it is $[0.5 * v, 1.5 * v]$ 12.35.3.4 `bool QwtScaleEngine::contains (const QwtDoubleInterval & interval, double value) const` `[protected, inherited]`

Check if an interval "contains" a value

Parameters

<i>interval</i>	Interval
<i>value</i>	Value

See also

[QwtScaleArithmetic::compareEps\(\)](#)

12.35.3.5 `double QwtScaleEngine::divideInterval (double intervalSize, int numSteps) const` **[protected, inherited]**

Calculate a step size for an interval size

Parameters

<i>intervalSize</i>	Interval size
<i>numSteps</i>	Number of steps

Returns

Step size

12.35.3.6 `QwtScaleDiv QwtLog10ScaleEngine::divideScale (double x1, double x2, int maxMajSteps, int maxMinSteps, double stepSize = 0.0) const` **[virtual]**

Calculate a scale division.

Parameters

<i>x1</i>	First interval limit
<i>x2</i>	Second interval limit
<i>maxMajSteps</i>	Maximum for the number of major steps
<i>maxMinSteps</i>	Maximum number of minor steps
<i>stepSize</i>	Step size. If <i>stepSize</i> == 0, the <i>scaleEngine</i> calculates one.

See also

`QwtScaleEngine::stepSize()`, `QwtLog10ScaleEngine::subDivide()`

Implements [QwtScaleEngine](#).

12.35.3.7 `QwtDoubleInterval QwtLog10ScaleEngine::log10 (const QwtDoubleInterval & interval) const` **[protected]**

Return the interval `[log10(interval.minValue(), log10(interval.maxValue)]`

12.35.3.8 double QwtScaleEngine::lowerMargin () const [inherited]**Returns**

the margin at the lower end of the scale The default margin is 0.

See also

[setMargins\(\)](#)

12.35.3.9 QwtDoubleInterval QwtLog10ScaleEngine::pow10 (const QwtDoubleInterval & *interval*) const [protected]

Return the interval [pow10(interval.minValue(), pow10(interval.maxValue)]

12.35.3.10 double QwtScaleEngine::reference () const [inherited]**Returns**

the reference value

See also

[setReference\(\)](#), [setAttribute\(\)](#)

12.35.3.11 void QwtScaleEngine::setAttribute (Attribute *attribute*, bool *on* = true) [inherited]

Change a scale attribute

Parameters

<i>attribute</i>	Attribute to change
<i>on</i>	On/Off

See also

[Attribute](#), [testAttribute\(\)](#)

12.35.3.12 void QwtScaleEngine::setAttributes (int *attributes*) [inherited]

Change the scale attribute

Parameters

<i>attributes</i>	Set scale attributes
-------------------	----------------------

See also

[Attribute](#), [attributes\(\)](#)

12.35.3.13 void QwtScaleEngine::setMargins (double *lower*, double *upper*) [*inherited*]

Specify margins at the scale's endpoints.

Parameters

<i>lower</i>	minimum distance between the scale's lower boundary and the smallest enclosed value
<i>upper</i>	minimum distance between the scale's upper boundary and the greatest enclosed value

Margins can be used to leave a minimum amount of space between the enclosed intervals and the boundaries of the scale.

Warning

- [QwtLog10ScaleEngine](#) measures the margins in decades.

See also

[upperMargin\(\)](#), [lowerMargin\(\)](#)

12.35.3.14 void QwtScaleEngine::setReference (double *r*) [inherited]

Specify a reference point.

Parameters

<i>r</i>	new reference value
----------	---------------------

The reference point is needed if options IncludeReference or Symmetric are active. Its default value is 0.0.

See also

[Attribute](#)

12.35.3.15 `QwtValueList QwtScaleEngine::strip (const QwtValueList & ticks,
const QwtDoubleInterval & interval) const` `[protected,
inherited]`

Remove ticks from a list, that are not inside an interval

Parameters

<i>ticks</i>	Tick list
<i>interval</i>	Interval

Returns

Stripped tick list

12.35.3.16 `bool QwtScaleEngine::testAttribute (Attribute attribute) const`
`[inherited]`

Check if a attribute is set.

Parameters

<i>attribute</i>	Attribute to be tested
------------------	------------------------

See also

[Attribute](#), [setAttribute\(\)](#)

12.35.3.17 `QwtScaleTransformation * QwtLog10ScaleEngine::transformation () const` `[virtual]`

Return a transformation, for logarithmic (base 10) scales

Implements [QwtScaleEngine](#).

12.35.3.18 `double QwtScaleEngine::upperMargin () const` `[inherited]`

Returns

the margin at the upper end of the scale The default margin is 0.

See also

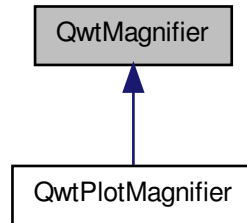
[setMargins\(\)](#)

12.36 QwtMagnifier Class Reference

[QwtMagnifier](#) provides zooming, by magnifying in steps.

```
#include <qwt_magnifier.h>
```

Inheritance diagram for QwtMagnifier:



Public Member Functions

- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [getMouseButton](#) (int &button, int &buttonState) const
- void [getZoomInKey](#) (int &key, int &modifiers) const
- void [getZoomOutKey](#) (int &key, int &modifiers) const
- bool [isEnabled](#) () const
- double [keyFactor](#) () const
- double [mouseFactor](#) () const
- const QWidget * [parentWidget](#) () const
- QWidget * [parentWidget](#) ()
- [QwtMagnifier](#) (QWidget *)
- void [setEnabled](#) (bool)
- void [setKeyFactor](#) (double)
- void [setMouseButton](#) (int button, int buttonState=Qt::NoButton)
- void [setMouseFactor](#) (double)
- void [setWheelButtonState](#) (int buttonState)
- void [setWheelFactor](#) (double)
- void [setZoomInKey](#) (int key, int modifiers)
- void [setZoomOutKey](#) (int key, int modifiers)
- int [wheelButtonState](#) () const
- double [wheelFactor](#) () const
- virtual [~QwtMagnifier](#) ()

Protected Member Functions

- virtual void [rescale](#) (double factor)=0
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)
- virtual void [widgetWheelEvent](#) (QWheelEvent *)

12.36.1 Detailed Description

[QwtMagnifier](#) provides zooming, by magnifying in steps. Using [QwtMagnifier](#) a plot can be zoomed in/out in steps using keys, the mouse wheel or moving a mouse button in vertical direction.

12.36.2 Constructor & Destructor Documentation**12.36.2.1 QwtMagnifier::QwtMagnifier (QWidget * *parent*) [explicit]**

Constructor

Parameters

<i>parent</i>	Widget to be magnified
---------------	------------------------

12.36.2.2 QwtMagnifier::~~QwtMagnifier () [virtual]

Destructor.

12.36.3 Member Function Documentation**12.36.3.1 bool QwtMagnifier::eventFilter (QObject * *o*, QEvent * *e*) [virtual]**

Event filter.

When [isEnabled\(\)](#) the mouse events of the observed widget are filtered.

See also

[widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#) [widgetKeyReleaseEvent\(\)](#)

12.36.3.2 void QwtMagnifier::getMouseButton (int & *button*, int & *buttonState*) const

See also

[setMouseButton\(\)](#)

12.36.3.3 void QwtMagnifier::getZoomInKey (int & *key*, int & *modifiers*) const

See also

[setZoomInKey\(\)](#)

12.36.3.4 void QwtMagnifier::getZoomOutKey (int & *key*, int & *modifiers*) const

See also

[setZoomOutKey\(\)](#)

12.36.3.5 bool QwtMagnifier::isEnabled () const

Returns

true when enabled, false otherwise

See also

[setEnabled\(\)](#), [eventFilter\(\)](#)

12.36.3.6 double QwtMagnifier::keyFactor () const

Returns

Key factor

See also

[setKeyFactor\(\)](#)

12.36.3.7 double QwtMagnifier::mouseFactor () const**Returns**

Mouse factor

See also

[setMouseFactor\(\)](#)

12.36.3.8 QWidget * QwtMagnifier::parentWidget ()**Returns**

Parent widget, where the rescaling happens

12.36.3.9 const QWidget * QwtMagnifier::parentWidget () const**Returns**

Parent widget, where the rescaling happens

**12.36.3.10 virtual void QwtMagnifier::rescale (double *factor*)
[protected, pure virtual]**

Rescale the parent widget

Parameters

<i>factor</i>	Scale factor
---------------	--------------

Implemented in [QwtPlotMagnifier](#).

12.36.3.11 void QwtMagnifier::setEnabled (bool *on*)

En/disable the magnifier.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>on</i>	true or false
-----------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.36.3.12 void QwtMagnifier::setKeyFactor (double *factor*)

Change the key factor.

The key factor defines the ratio between the current range on the parent widget and the zoomed range for each key press of the zoom in/out keys. The default value is 0.9.

Parameters

<i>factor</i>	Key factor
---------------	------------

See also

[keyFactor\(\)](#), [setZoomInKey\(\)](#), [setZoomOutKey\(\)](#), [setWheelFactor](#), [setMouseFactor\(\)](#)

12.36.3.13 void QwtMagnifier::setMouseButton (int *button*, int *buttonState* = *Qt::NoButton*)

Assign the mouse button, that is used for zooming in/out. The default value is Qt::RightButton.

Parameters

<i>button</i>	Button
<i>buttonState</i>	Button state

See also

[getMouseButton\(\)](#)

12.36.3.14 void QwtMagnifier::setMouseFactor (double *factor*)

Change the mouse factor.

The mouse factor defines the ratio between the current range on the parent widget and the zoomed range for each vertical mouse movement. The default value is 0.95.

Parameters

<i>factor</i>	Wheel factor
---------------	--------------

See also

[mouseFactor\(\)](#), [setMouseButton\(\)](#), [setWheelFactor\(\)](#), [setKeyFactor\(\)](#)

12.36.3.15 void QwtMagnifier::setWheelButtonState (int *buttonState*)

Assign a mandatory button state for zooming in/out using the wheel. The default button state is Qt::NoButton.

Parameters

<i>buttonState</i>	Button state
--------------------	--------------

See also

[wheelButtonState\(\)](#)

12.36.3.16 void QwtMagnifier::setWheelFactor (double *factor*)

Change the wheel factor.

The wheel factor defines the ratio between the current range on the parent widget and the zoomed range for each step of the wheel. The default value is 0.9.

Parameters

<i>factor</i>	Wheel factor
---------------	--------------

See also

[wheelFactor\(\)](#), [setWheelButtonState\(\)](#), [setMouseFactor\(\)](#), [setKeyFactor\(\)](#)

12.36.3.17 void QwtMagnifier::setZoomInKey (int *key*, int *modifiers*)

Assign the key, that is used for zooming in. The default combination is Qt::Key_Plus + Qt::NoModifier.

Parameters

<i>key</i>	
<i>modifiers</i>	

See also

[getZoomInKey\(\)](#), [setZoomOutKey\(\)](#)

12.36.3.18 void QwtMagnifier::setZoomOutKey (int *key*, int *modifiers*)

Assign the key, that is used for zooming out. The default combination is Qt::Key_Minus + Qt::NoModifier.

Parameters

<i>key</i>	
<i>modifiers</i>	

See also

[getZoomOutKey\(\)](#), [setZoomOutKey\(\)](#)

12.36.3.19 int QwtMagnifier::wheelButtonState () const

Returns

Wheel button state

See also

[setWheelButtonState\(\)](#)

12.36.3.20 double QwtMagnifier::wheelFactor () const

Returns

Wheel factor

See also

[setWheelFactor\(\)](#)

12.36.3.21 void QwtMagnifier::widgetKeyPressEvent (QKeyEvent * *ke*) [protected, virtual]

Handle a key press event for the observed widget.

Parameters

<i>ke</i>	Key event
-----------	-----------

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.36.3.22 `void QwtMagnifier::widgetKeyReleaseEvent (QKeyEvent *)`
[protected, virtual]

Handle a key release event for the observed widget.

Parameters

<i>ke</i>	Key event
-----------	-----------

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.36.3.23 `void QwtMagnifier::widgetMouseMoveEvent (QMouseEvent * me`
`) [protected, virtual]`

Handle a mouse move event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#),

12.36.3.24 `void QwtMagnifier::widgetMousePressEvent (QMouseEvent * me`
`) [protected, virtual]`

Handle a mouse press event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#)

12.36.3.25 void QwtMagnifier::widgetMouseEvent (QMouseEvent *) [protected, virtual]

Handle a mouse release event for the observed widget.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#),

12.36.3.26 void QwtMagnifier::widgetWheelEvent (QWheelEvent * we) [protected, virtual]

Handle a wheel event for the observed widget.

Parameters

<i>we</i>	Wheel event
-----------	-------------

See also

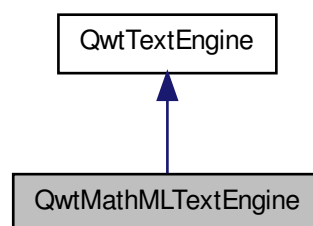
[eventFilter\(\)](#)

12.37 QwtMathMLTextEngine Class Reference

Text Engine for the MathML renderer of the Qt solutions package.

```
#include <qwt_mathml_text_engine.h>
```

Inheritance diagram for QwtMathMLTextEngine:



Public Member Functions

- virtual void [draw](#) (QPainter *painter, const QRect &rect, int flags, const QString &text) const

- virtual int [heightForWidth](#) (const QFont &font, int flags, const QString &text, int width) const
- virtual bool [mightRender](#) (const QString &) const
- [QwtMathMLTextEngine](#) ()
- virtual void [textMargins](#) (const QFont &, const QString &, int &left, int &right, int &top, int &bottom) const
- virtual QSize [textSize](#) (const QFont &font, int flags, const QString &text) const
- virtual [~QwtMathMLTextEngine](#) ()

12.37.1 Detailed Description

Text Engine for the MathML renderer of the Qt solutions package. The Qt Solution package includes a renderer for MathML <http://www.trolltech.com/products/qt/addon/solutions> that is available for owners of a commercial Qt license. You need a version ≥ 2.1 , that is only available for Qt4.

To enable MathML support the following code needs to be added to the application:

```
#include <qwt_mathml_text_engine.h>

QwtText::setTextEngine(QwtText::MathMLText, new QwtMathMLTextEngine());
```

See also

[QwtTextEngine](#), [QwtText::setTextEngine](#)

Warning

Unfortunately the MathML renderer doesn't support rotating of texts.

12.37.2 Constructor & Destructor Documentation

12.37.2.1 QwtMathMLTextEngine::QwtMathMLTextEngine ()

Constructor.

12.37.2.2 QwtMathMLTextEngine::~QwtMathMLTextEngine () [virtual]

Destructor.

12.37.3 Member Function Documentation

12.37.3.1 `void QwtMathMLTextEngine::draw (QPainter * painter, const QRect & rect, int flags, const QString & text) const` **[virtual]**

Draw the text in a clipping rectangle

Parameters

<i>painter</i>	Painter
<i>rect</i>	Clipping rectangle
<i>flags</i>	Bitwise OR of the flags like in for QPainter::drawText
<i>text</i>	Text to be rendered

Implements [QwtTextEngine](#).

12.37.3.2 `int QwtMathMLTextEngine::heightForWidth (const QFont & font, int flags, const QString & text, int width) const` **[virtual]**

Find the height for a given width

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered
<i>width</i>	Width

Returns

Calculated height

Implements [QwtTextEngine](#).

12.37.3.3 `bool QwtMathMLTextEngine::mightRender (const QString & text) const` **[virtual]**

Test if a string can be rendered by [QwtMathMLTextEngine](#)

Parameters

<i>text</i>	Text to be tested
-------------	-------------------

Returns

true, if text begins with "<math>".

Implements [QwtTextEngine](#).

12.37.3.4 void QwtMathMLTextEngine::textMargins (const QFont &, const QString &, int & *left*, int & *right*, int & *top*, int & *bottom*) const [virtual]

Return margins around the texts

Parameters

<i>left</i>	Return 0
<i>right</i>	Return 0
<i>top</i>	Return 0
<i>bottom</i>	Return 0

Implements [QwtTextEngine](#).

12.37.3.5 QSize QwtMathMLTextEngine::textSize (const QFont & *font*, int *flags*, const QString & *text*) const [virtual]

Returns the size, that is needed to render text

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered

Returns

Caluclated size

Implements [QwtTextEngine](#).

12.38 QwtMetricsMap Class Reference

A Map to translate between layout, screen and paint device metrics.

```
#include <qwt_layout_metrics.h>
```

Public Member Functions

- QPoint **deviceToLayout** (const QPoint &, const QPainter *=*NULL*) const
- QSize **deviceToLayout** (const QSize &) const
- QRect **deviceToLayout** (const QRect &, const QPainter *=*NULL*) const
- QwtPolygon **deviceToLayout** (const QwtPolygon &, const QPainter *=*NULL*) const
- int **deviceToLayoutX** (int *x*) const
- int **deviceToLayoutY** (int *y*) const
- bool **isIdentity** () const

- QSize **layoutToDevice** (const QSize &) const
- QwtPolygon **layoutToDevice** (const QwtPolygon &, const QPainter *=*NULL*) const
- QRect **layoutToDevice** (const QRect &, const QPainter *=*NULL*) const
- QPoint **layoutToDevice** (const QPoint &, const QPainter *=*NULL*) const
- int **layoutToDeviceX** (int x) const
- int **layoutToDeviceY** (int y) const
- QSize **layoutToScreen** (const QSize &) const
- QRect **layoutToScreen** (const QRect &) const
- QPoint **layoutToScreen** (const QPoint &point) const
- int **layoutToScreenX** (int x) const
- int **layoutToScreenY** (int y) const
- QPoint **screenToLayout** (const QPoint &) const
- QRect **screenToLayout** (const QRect &) const
- QSize **screenToLayout** (const QSize &) const
- int **screenToLayoutX** (int x) const
- int **screenToLayoutY** (int y) const
- void **setMetrics** (const QPaintDevice *layoutMetrics, const QPaintDevice *deviceMetrics)

Static Public Member Functions

- static QwtPolygon [translate](#) (const QMatrix &, const QwtPolygon &)
- static QRect [translate](#) (const QMatrix &, const QRect &)

12.38.1 Detailed Description

A Map to translate between layout, screen and paint device metrics. Qt3 supports painting in integer coordinates only. Therefore it is not possible to scale the layout in screen coordinates to layouts in higher resolutions (f.e printing) without losing the higher precision. [QwtMetricsMap](#) is used to incorporate the various widget attributes (always in screen resolution) into the layout/printing code of [QwtPlot](#).

Qt4 is able to paint floating point based coordinates, what makes it possible always to render in screen coordinates (with a common scale factor). [QwtMetricsMap](#) will be obsolete as soon as Qt3 support has been dropped (Qwt 6.x).

12.38.2 Member Function Documentation

12.38.2.1 QwtPolygon QwtMetricsMap::translate (const QMatrix & *m*, const QwtPolygon & *pa*) [static]

Wrapper for QMatrix::map.

Parameters

<i>m</i>	Matrix
<i>pa</i>	Polygon to translate

Returns

Translated polygon

12.38.2.2 QRect QwtMetricsMap::translate (const QMatrix & *m*, const QRect & *rect*) [static]

Wrapper for QMatrix::mapRect.

Parameters

<i>m</i>	Matrix
<i>rect</i>	Rectangle to translate

Returns

Translated rectangle

12.39 QwtPainter Class Reference

A collection of QPainter workarounds.

```
#include <qwt_painter.h>
```

Static Public Member Functions

- static bool [deviceClipping](#) ()
- static const QRect & [deviceClipRect](#) ()
- static void [drawColorBar](#) (QPainter *painter, const [QwtColorMap](#) &, const [QwtDoubleInterval](#) &, const [QwtScaleMap](#) &, Qt::Orientation, const QRect &)
- static void [drawEllipse](#) (QPainter *, const QRect &)
- static void [drawFocusRect](#) (QPainter *, QWidget *)
- static void [drawFocusRect](#) (QPainter *, QWidget *, const QRect &)
- static void [drawLine](#) (QPainter *, int x1, int y1, int x2, int y2)
- static void [drawLine](#) (QPainter *, const QPoint &p1, const QPoint &p2)
- static void [drawPie](#) (QPainter *, const QRect &r, int a, int alen)
- static void [drawPoint](#) (QPainter *, int x, int y)
- static void [drawPolygon](#) (QPainter *, const QwtPolygon &pa)
- static void [drawPolyline](#) (QPainter *, const QwtPolygon &pa)
- static void [drawRect](#) (QPainter *, int x, int y, int w, int h)
- static void [drawRect](#) (QPainter *, const QRect &rect)
- static void [drawRoundFrame](#) (QPainter *, const QRect &, int width, const QPalette &, bool sunken)
- static void [drawSimpleRichText](#) (QPainter *, const QRect &, int flags, QTextDocument &)
- static void [drawText](#) (QPainter *, int x, int y, int w, int h, int flags, const QString &)
- static void [drawText](#) (QPainter *, int x, int y, const QString &)

- static void [drawText](#) (QPainter *, const QPoint &, const QString &)
- static void [drawText](#) (QPainter *, const QRect &, int flags, const QString &)
- static void [fillRect](#) (QPainter *, const QRect &, const QBrush &)
- static const [QwtMetricsMap](#) & [metricsMap](#) ()
- static void [resetMetricsMap](#) ()
- static QPen [scaledPen](#) (const QPen &)
- static void [setClipRect](#) (QPainter *, const QRect &)
- static void [setDeviceClipping](#) (bool)
- static void [setMetricsMap](#) (const [QwtMetricsMap](#) &)
- static void [setMetricsMap](#) (const QPaintDevice *layout, const QPaintDevice *device)

12.39.1 Detailed Description

A collection of QPainter workarounds. 1) Clipping to coordinate system limits (Qt3 only)

On X11 pixel coordinates are stored in shorts. Qt produces overruns when mapping QCOORDS to shorts.

2) Scaling to device metrics

QPainter scales fonts, line and fill patterns to the metrics of the paint device. Other values like the geometries of rects, points remain device independent. To enable a device independent widget implementation, [QwtPainter](#) adds scaling of these geometries. (Unfortunately QPainter::scale scales both types of paintings, so the objects of the first type would be scaled twice).

12.39.2 Member Function Documentation

12.39.2.1 bool QwtPainter::deviceClipping () [[inline](#), [static](#)]

Returns whether device clipping is enabled. On X11 the default is enabled, otherwise it is disabled.

See also

[QwtPainter::setDeviceClipping\(\)](#)

12.39.2.2 const QRect & QwtPainter::deviceClipRect () [[static](#)]

Returns rect for device clipping

See also

[QwtPainter::setDeviceClipping\(\)](#)

12.39.2.3 void QwtPainter::drawEllipse (QPainter * *painter*, const QRect & *rect*) [static]

Wrapper for QPainter::drawEllipse()

12.39.2.4 void QwtPainter::drawLine (QPainter * *painter*, int *x1*, int *y1*, int *x2*, int *y2*) [static]

Wrapper for QPainter::drawLine()

12.39.2.5 void QwtPainter::drawLine (QPainter * *painter*, const QPoint & *p1*, const QPoint & *p2*) [inline, static]

Wrapper for QPainter::drawLine()

12.39.2.6 void QwtPainter::drawPie (QPainter * *painter*, const QRect & *rect*, int *a*, int *alen*) [static]

Wrapper for QPainter::drawPie()

12.39.2.7 void QwtPainter::drawPoint (QPainter * *painter*, int *x*, int *y*) [static]

Wrapper for QPainter::drawPoint()

12.39.2.8 void QwtPainter::drawPolygon (QPainter * *painter*, const QwtPolygon & *pa*) [static]

Wrapper for QPainter::drawPolygon()

12.39.2.9 void QwtPainter::drawPolyline (QPainter * *painter*, const QwtPolygon & *pa*) [static]

Wrapper for QPainter::drawPolyline()

12.39.2.10 void QwtPainter::drawRect (QPainter * *painter*, int *x*, int *y*, int *w*, int *h*) [static]

Wrapper for QPainter::drawRect()

12.39.2.11 void QwtPainter::drawRect (QPainter * *painter*, const QRect & *rect*) [static]

Wrapper for QPainter::drawRect()

12.39.2.12 void QwtPainter::drawRoundFrame (QPainter * *painter*, const QRect & *rect*, int *width*, const QPalette & *palette*, bool *sunken*) [static]

Draw a round frame.

12.39.2.13 void QwtPainter::drawSimpleRichText (QPainter * *painter*, const QRect & *rect*, int *flags*, QTextDocument & *text*) [static]

Wrapper for QSimpleRichText::draw()

12.39.2.14 void QwtPainter::drawText (QPainter * *painter*, int *x*, int *y*, int *w*, int *h*, int *flags*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.15 void QwtPainter::drawText (QPainter * *painter*, int *x*, int *y*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.16 void QwtPainter::drawText (QPainter * *painter*, const QPoint & *pos*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.17 void QwtPainter::drawText (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.18 void QwtPainter::fillRect (QPainter * *painter*, const QRect & *rect*, const QBrush & *brush*) [static]

Wrapper for QPainter::fillRect()

12.39.2.19 `const QwtMetricsMap & QwtPainter::metricsMap () [static]`**Returns**

Metrics map

12.39.2.20 `void QwtPainter::resetMetricsMap () [static]`

Reset the metrics map to the ratio 1:1

See also

[QwtPainter::setMetricsMap\(\)](#), [QwtPainter::resetMetricsMap\(\)](#)

12.39.2.21 `QPen QwtPainter::scaledPen (const QPen & pen) [static]`

Scale a pen according to the layout metrics.

The width of non cosmetic pens is scaled from screen to layout metrics, so that they look similar on paint devices with different resolutions.

Parameters

<i>pen</i>	Unscaled pen
------------	--------------

Returns

Scaled pen

12.39.2.22 `void QwtPainter::setClipRect (QPainter * painter, const QRect & rect) [static]`

Wrapper for QPainter::setClipRect()

12.39.2.23 `void QwtPainter::setDeviceClipping (bool enable) [static]`

En/Disable device clipping.

On X11 the default for device clipping is enabled, otherwise it is disabled.

See also

[QwtPainter::deviceClipping\(\)](#)

12.39.2.24 `void QwtPainter::setMetricsMap (const QwtMetricsMap & map)`
[static]

Change the metrics map

See also

[QwtPainter::resetMetricsMap\(\)](#), [QwtPainter::metricsMap\(\)](#)

12.39.2.25 `void QwtPainter::setMetricsMap (const QPaintDevice * layout,
const QPaintDevice * device)` **[static]**

Scale all [QwtPainter](#) drawing operations using the ratio $\text{QwtPaintMetrics}(\text{from}).\text{logicalDpiX}() / \text{QwtPaintMetrics}(\text{to}).\text{logicalDpiX}()$ and $\text{QwtPaintMetrics}(\text{from}).\text{logicalDpiY}() / \text{QwtPaintMetrics}(\text{to}).\text{logicalDpiY}()$

See also

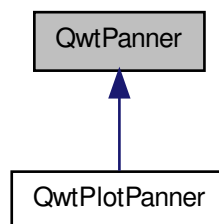
[QwtPainter::resetScaleMetrics\(\)](#), [QwtPainter::scaleMetricsX\(\)](#), [QwtPainter::scaleMetricsY\(\)](#)

12.40 QwtPanner Class Reference

[QwtPanner](#) provides panning of a widget.

```
#include <qwt_panner.h>
```

Inheritance diagram for QwtPanner:



Signals

- void [moved](#) (int dx, int dy)
- void [panned](#) (int dx, int dy)

Public Member Functions

- const [QCursor](#) [cursor](#) () const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [getAbortKey](#) (int &key, int &state) const
- void [getMouseButton](#) (int &button, int &buttonState) const
- bool [isEnabled](#) () const
- bool [isOrientationEnabled](#) (Qt::Orientation) const
- Qt::Orientations [orientations](#) () const
- [QwtPanner](#) (QWidget *parent)
- void [setAbortKey](#) (int key, int state=Qt::NoButton)
- void [setCursor](#) (const QCursor &)
- void [setEnabled](#) (bool)
- void [setMouseButton](#) (int button, int buttonState=Qt::NoButton)
- void [setOrientations](#) (Qt::Orientations)
- virtual [~QwtPanner](#) ()

Protected Member Functions

- virtual void [paintEvent](#) (QPaintEvent *)
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)

12.40.1 Detailed Description

[QwtPanner](#) provides panning of a widget. [QwtPanner](#) grabs the contents of a widget, that can be dragged in all directions. The offset between the start and the end position is emitted by the panned signal.

[QwtPanner](#) grabs the content of the widget into a pixmap and moves the pixmap around, without initiating any repaint events for the widget. Areas, that are not part of content are not painted while panning in process. This makes panning fast enough for widgets, where repaints are too slow for mouse movements.

For widgets, where repaints are very fast it might be better to implement panning manually by mapping mouse events into paint events.

12.40.2 Constructor & Destructor Documentation**12.40.2.1 QwtPanner::QwtPanner (QWidget *parent)**

Creates an panner that is enabled for the left mouse button.

Parameters

<i>parent</i>	Parent widget to be panned
---------------	----------------------------

12.40.2.2 QwtPanner::~~QwtPanner () [virtual]

Destructor.

12.40.3 Member Function Documentation

12.40.3.1 const QCursor QwtPanner::cursor () const

Returns

Cursor that is active while panning

See also

[setCursor\(\)](#)

12.40.3.2 bool QwtPanner::eventFilter (QObject * o, QEvent * e) [virtual]

Event filter.

When [isEnabled\(\)](#) the mouse events of the observed widget are filtered.

See also

[widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#)

12.40.3.3 void QwtPanner::getAbortKey (int & key, int & state) const

Get the abort key.

12.40.3.4 void QwtPanner::getMouseButton (int & button, int & buttonState) const

Get the mouse button.

12.40.3.5 bool QwtPanner::isEnabled () const**Returns**

true when enabled, false otherwise

See also

[setEnabled](#), [eventFilter\(\)](#)

12.40.3.6 bool QwtPanner::isOrientationEnabled (Qt::Orientation o) const

Return true if a orientatio is enabled

See also

[orientations\(\)](#), [setOrientations\(\)](#)

12.40.3.7 void QwtPanner::moved (int dx, int dy) [signal]

Signal emitted, while the widget moved, but panning is not finished.

Parameters

<i>dx</i>	Offset in horizontal direction
<i>dy</i>	Offset in vertical direction

12.40.3.8 Qt::Orientations QwtPanner::orientations () const

Return the orientation, where paning is enabled.

12.40.3.9 void QwtPanner::paintEvent (QPaintEvent *pe) [protected, virtual]

Paint event.

Repaint the grabbed pixmap on its current position and fill the empty spaces by the background of the parent widget.

Parameters

<i>pe</i>	Paint event
-----------	-------------

12.40.3.10 void QwtPanner::panned (int *dx*, int *dy*) [signal]

Signal emitted, when panning is done

Parameters

<i>dx</i>	Offset in horizontal direction
<i>dy</i>	Offset in vertical direction

12.40.3.11 void QwtPanner::setAbortKey (int *key*, int *state* = *Qt::NoButton*)

Change the abort key The defaults are Qt::Key_Escape and Qt::NoButton

Parameters

<i>key</i>	Key (See Qt::Keycode)
<i>state</i>	State

12.40.3.12 void QwtPanner::setCursor (const QCursor & *cursor*)

Change the cursor, that is active while panning The default is the cursor of the parent widget.

Parameters

<i>cursor</i>	New cursor
---------------	------------

See also

[setCursor\(\)](#)

12.40.3.13 void QwtPanner::setEnabled (bool *on*)

En/disable the panner.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>on</i>	true or false
-----------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.40.3.14 `void QwtPanner::setMouseButton (int button, int buttonState = Qt::NoButton)`

Change the mouse button The defaults are Qt::LeftButton and Qt::NoButton

12.40.3.15 `void QwtPanner::setOrientations (Qt::Orientations o)`

Set the orientations, where panning is enabled The default value is in both directions:
Qt::Horizontal | Qt::Vertical

/param o Orientation

12.40.3.16 `void QwtPanner::widgetKeyPressEvent (QKeyEvent * ke)`
[protected, virtual]

Handle a key press event for the observed widget.

Parameters

<i>ke</i>	Key event
-----------	-----------

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.40.3.17 `void QwtPanner::widgetKeyReleaseEvent (QKeyEvent *)`
[protected, virtual]

Handle a key release event for the observed widget.

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.40.3.18 `void QwtPanner::widgetMouseMoveEvent (QMouseEvent * me)`
[protected, virtual]

Handle a mouse move event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#)

12.40.3.19 void QwtPanner::widgetMouseEvent (QMouseEvent * *me*)
[protected, virtual]

Handle a mouse press event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#),

12.40.3.20 void QwtPanner::widgetMouseEvent (QMouseEvent * *me*)
[protected, virtual]

Handle a mouse release event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

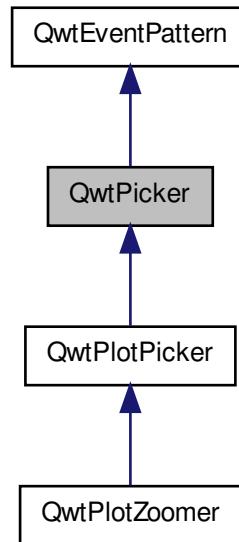
[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#),

12.41 QwtPicker Class Reference

[QwtPicker](#) provides selections on a widget.

```
#include <qwt_picker.h>
```

Inheritance diagram for QwtPicker:



Public Types

- enum [DisplayMode](#) {
 AlwaysOff,
 AlwaysOn,
 ActiveOnly }
- enum [KeyPatternCode](#) {
 KeySelect1,
 KeySelect2,
 KeyAbort,
 KeyLeft,
 KeyRight,
 KeyUp,
 KeyDown,
 KeyRedo,
 KeyUndo,
 KeyHome,
 KeyPatternCount }

- enum [MousePatternCode](#) {
 MouseSelect1,
 MouseSelect2,
 MouseSelect3,
 MouseSelect4,
 MouseSelect5,
 MouseSelect6,
 MousePatternCount }
- enum [RectSelectionType](#) {
 CornerToCorner = 64,
 CenterToCorner = 128,
 CenterToRadius = 256 }
- enum [ResizeMode](#) {
 Stretch,
 KeepSize }
- enum [RubberBand](#) {
 NoRubberBand = 0,
 HLineRubberBand,
 VLineRubberBand,
 CrossRubberBand,
 RectRubberBand,
 EllipseRubberBand,
 PolygonRubberBand,
 UserRubberBand = 100 }
- enum [SelectionMode](#) {
 ClickSelection = 1024,
 DragSelection = 2048 }
- enum [SelectionType](#) {
 NoSelection = 0,
 PointSelection = 1,
 RectSelection = 2,
 PolygonSelection = 4 }

Signals

- void [appended](#) (const QPoint &pos)
- void [changed](#) (const QwtPolygon &pa)
- void [moved](#) (const QPoint &pos)
- void [selected](#) (const QwtPolygon &pa)

Public Member Functions

- virtual void [drawRubberBand](#) (QPainter *) const
- virtual void [drawTracker](#) (QPainter *) const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [initKeyPattern](#) ()
- void [initMousePattern](#) (int numButtons)
- bool [isActive](#) () const
- bool [isEnabled](#) () const
- bool [keyMatch](#) (uint pattern, const QKeyEvent *) const
- const QwtArray< [KeyPattern](#) > & [keyPattern](#) () const
- QwtArray< [KeyPattern](#) > & [keyPattern](#) ()
- bool [mouseMatch](#) (uint pattern, const QMouseEvent *) const
- const QwtArray< [MousePattern](#) > & [mousePattern](#) () const
- QwtArray< [MousePattern](#) > & [mousePattern](#) ()
- QWidget * [parentWidget](#) ()
- const QWidget * [parentWidget](#) () const
- virtual QRect [pickRect](#) () const
- [QwtPicker](#) (int selectionFlags, [RubberBand](#) rubberBand, [DisplayMode](#) trackerMode, QWidget *)
- [QwtPicker](#) (QWidget *parent)
- [ResizeMode](#) [resizeMode](#) () const
- [RubberBand](#) [rubberBand](#) () const
- QPen [rubberBandPen](#) () const
- const QwtPolygon & [selection](#) () const
- int [selectionFlags](#) () const
- virtual void [setEnabled](#) (bool)
- void [setKeyPattern](#) (uint pattern, int key, int state=Qt::NoButton)
- void [setKeyPattern](#) (const QwtArray< [KeyPattern](#) > &)
- void [setMousePattern](#) (uint pattern, int button, int state=Qt::NoButton)
- void [setMousePattern](#) (const QwtArray< [MousePattern](#) > &)
- virtual void [setResizeMode](#) ([ResizeMode](#))
- virtual void [setRubberBand](#) ([RubberBand](#))
- virtual void [setRubberBandPen](#) (const QPen &)
- virtual void [setSelectionFlags](#) (int)
- virtual void [setTrackerFont](#) (const QFont &)
- virtual void [setTrackerMode](#) ([DisplayMode](#))
- virtual void [setTrackerPen](#) (const QPen &)
- QFont [trackerFont](#) () const
- [DisplayMode](#) [trackerMode](#) () const
- QPen [trackerPen](#) () const
- QPoint [trackerPosition](#) () const
- QRect [trackerRect](#) (const QFont &) const
- virtual [QwtText](#) [trackerText](#) (const QPoint &pos) const
- virtual [~QwtPicker](#) ()

Protected Member Functions

- virtual bool [accept](#) (QwtPolygon &selection) const
- virtual void [append](#) (const QPoint &)
- virtual void [begin](#) ()
- virtual bool [end](#) (bool ok=true)
- virtual bool [keyMatch](#) (const [KeyPattern](#) &, const QKeyEvent *) const
- virtual bool [mouseMatch](#) (const [MousePattern](#) &, const QMouseEvent *) const
- virtual void [move](#) (const QPoint &)
- virtual void [reset](#) ()
- const QWidget * [rubberBandWidget](#) () const
- virtual [QwtPickerMachine](#) * [stateMachine](#) (int) const
- virtual void [stretchSelection](#) (const QSize &oldSize, const QSize &newSize)
- const QWidget * [trackerWidget](#) () const
- virtual void [transition](#) (const QEvent *)
- virtual void [updateDisplay](#) ()
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetLeaveEvent](#) (QEvent *)
- virtual void [widgetMouseDoubleClickEvent](#) (QMouseEvent *)
- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)
- virtual void [widgetWheelEvent](#) (QWheelEvent *)

12.41.1 Detailed Description

[QwtPicker](#) provides selections on a widget. [QwtPicker](#) filters all mouse and keyboard events of a widget and translates them into an array of selected points. Depending on the [QwtPicker::SelectionType](#) the selection might be a single point, a rectangle or a polygon. The selection process is supported by optional rubberbands (rubberband selection) and position trackers.

[QwtPicker](#) is useful for widgets where the event handlers can't be overloaded, like for components of composite widgets. It offers alternative handlers for mouse and key events.

Example

```
#include <qwt_picker.h>

QwtPicker *picker = new QwtPicker(widget);
picker->setTrackerMode(QwtPicker::ActiveOnly);
connect(picker, SIGNAL(selected(const QwtPolygon &)), ...);

// emit the position of clicks on widget
picker->setSelectionFlags(QwtPicker::PointSelection | QwtPicker::ClickSelection);

...

// now select rectangles
picker->setSelectionFlags(QwtPicker::RectSelection | QwtPicker::DragSelection);
picker->setRubberBand(QwtPicker::RectRubberBand);
```

The selection process uses the commands [begin\(\)](#), [append\(\)](#), [move\(\)](#) and [end\(\)](#). [append\(\)](#) adds a new point to the selection, [move\(\)](#) changes the position of the latest point.

The commands are initiated from a small state machine ([QwtPickerMachine](#)) that translates mouse and key events. There are a couple of predefined state machines for point, rect and polygon selections. The [selectionFlags\(\)](#) control which one should be used. It is possible to use other machines by overloading [stateMachine\(\)](#).

The picker is active ([isActive\(\)](#)), between [begin\(\)](#) and [end\(\)](#). In active state the rubber-band is displayed, and the tracker is visible in case of [trackerMode](#) is [ActiveOnly](#) or [AlwaysOn](#).

The cursor can be moved using the arrow keys. All selections can be aborted using the abort key. ([QwtEventPattern::KeyPatternCode](#))

Warning

In case of [QWidget::NoFocus](#) the focus policy of the observed widget is set to [QWidget::WheelFocus](#) and mouse tracking will be manipulated for [ClickSelection](#) while the picker is active, or if [trackerMode\(\)](#) is [AlwaysOn](#).

12.41.2 Member Enumeration Documentation

12.41.2.1 enum QwtPicker::DisplayMode

- [AlwaysOff](#)
Display never.
- [AlwaysOn](#)
Display always.
- [ActiveOnly](#)
Display only when the selection is active.

See also

[QwtPicker::setTrackerMode\(\)](#), [QwtPicker::trackerMode\(\)](#), [QwtPicker::isActive\(\)](#)

12.41.2.2 enum QwtEventPattern::KeyPatternCode [inherited]

Symbolic keyboard input codes.

Default initialization:

- [KeySelect1](#)
[Qt::Key_Return](#)

- KeySelect2
Qt::Key_Space
- KeyAbort
Qt::Key_Escape
- KeyLeft
Qt::Key_Left
- KeyRight
Qt::Key_Right
- KeyUp
Qt::Key_Up
- KeyDown
Qt::Key_Down
- KeyUndo
Qt::Key_Minus
- KeyRedo
Qt::Key_Plus
- KeyHome
Qt::Key_Escape

12.41.2.3 enum QwtEventPattern::MousePatternCode [inherited]

Symbolic mouse input codes.

The default initialization for 3 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::MidButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton

- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::MidButton + Qt::ShiftButton

The default initialization for 2 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

The default initialization for 1 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::LeftButton + Qt::ControlButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::LeftButton + Qt::ControlButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

See also

[initMousePattern\(\)](#)

12.41.2.4 enum QwtPicker::RectSelectionType

Selection subtype for RectSelection This enum type describes the type of rectangle selections. It can be or'd with [QwtPicker::RectSelectionType](#) and [QwtPicker::SelectionMode](#) and passed to [QwtPicker::setSelectionFlags\(\)](#).

- CornerToCorner
The first and the second selected point are the corners of the rectangle.
- CenterToCorner
The first point is the center, the second a corner of the rectangle.
- CenterToRadius
The first point is the center of a quadrat, calculated by the maximum of the x- and y-distance.

The default value is CornerToCorner.

See also

[QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::selectionFlags\(\)](#)

12.41.2.5 enum QwtPicker::ResizeMode

Controls what to do with the selected points of an active selection when the observed widget is resized.

- Stretch
All points are scaled according to the new size,
- KeepSize
All points remain unchanged.

The default value is Stretch.

See also

[QwtPicker::setResizeMode\(\)](#), [QwtPicker::resize\(\)](#)

12.41.2.6 enum QwtPicker::RubberBand

Rubberband style

- NoRubberBand
No rubberband.

- `HLineRubberBand` & `PointSelection`
A horizontal line.
- `VLineRubberBand` & `PointSelection`
A vertical line.
- `CrossRubberBand` & `PointSelection`
A horizontal and a vertical line.
- `RectRubberBand` & `RectSelection`
A rectangle.
- `EllipseRubberBand` & `RectSelection`
An ellipse.
- `PolygonRubberBand` & `PolygonSelection`
A polygon.
- `UserRubberBand`
Values \geq `UserRubberBand` can be used to define additional rubber bands.

The default value is `NoRubberBand`.

See also

[QwtPicker::setRubberBand\(\)](#), [QwtPicker::rubberBand\(\)](#)

12.41.2.7 enum QwtPicker::SelectionMode

Values of this enum type or'd together with a `SelectionType` value identifies which state machine should be used for the selection.

The default value is `ClickSelection`.

See also

[stateMachine\(\)](#)

12.41.2.8 enum QwtPicker::SelectionType

This enum type describes the type of a selection. It can be or'd with [QwtPicker::RectSelectionType](#) and [QwtPicker::SelectionMode](#) and passed to [QwtPicker::setSelectionFlags\(\)](#)

- `NoSelection`
Selection is disabled. Note this is different to the disabled state, as you might have a tracker.

- **PointSelection**
Select a single point.
- **RectSelection**
Select a rectangle.
- **PolygonSelection**
Select a polygon.

The default value is NoSelection.

See also

[QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::selectionFlags\(\)](#)

12.41.3 Constructor & Destructor Documentation

12.41.3.1 QwtPicker::QwtPicker (QWidget * *parent*) [explicit]

Constructor

Creates an picker that is enabled, but where selection flag is set to NoSelection, rubberband and tracker are disabled.

Parameters

<i>parent</i>	Parent widget, that will be observed
---------------	--------------------------------------

12.41.3.2 QwtPicker::QwtPicker (int *selectionFlags*, RubberBand *rubberBand*, DisplayMode *trackerMode*, QWidget * *parent*) [explicit]

Constructor

Parameters

<i>selectionFlags</i>	Or'd value of SelectionType, RectSelectionType and SelectionMode
<i>rubberBand</i>	Rubberband style
<i>trackerMode</i>	Tracker mode
<i>parent</i>	Parent widget, that will be observed

12.41.3.3 QwtPicker::~QwtPicker () [virtual]

Destructor.

12.41.4 Member Function Documentation

12.41.4.1 **bool QwtPicker::accept (QwtPolygon & *selection*) const**
[protected, virtual]

Validate and fixup the selection.

Accepts all selections unmodified

Parameters

<i>selection</i>	Selection to validate and fixup
------------------	---------------------------------

Returns

true, when accepted, false otherwise

Reimplemented in [QwtPlotZoomer](#).

12.41.4.2 **void QwtPicker::append (const QPoint & *pos*)** [protected, virtual]

Append a point to the selection and update rubberband and tracker. The [appended\(\)](#) signal is emitted.

Parameters

<i>pos</i>	Additional point
------------	------------------

See also

[isActive\(\)](#), [begin\(\)](#), [end\(\)](#), [move\(\)](#), [appended\(\)](#)

Reimplemented in [QwtPlotPicker](#).

12.41.4.3 **void QwtPicker::appended (const QPoint & *pos*)** [signal]

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

Parameters

<i>pos</i>	Position of the appended point.
------------	---------------------------------

See also

[append\(\)](#), [moved\(\)](#)

12.41.4.4 void QwtPicker::begin () [protected, virtual]

Open a selection setting the state to active

See also

[isActive\(\)](#), [end\(\)](#), [append\(\)](#), [move\(\)](#)

Reimplemented in [QwtPlotZoomer](#).

12.41.4.5 void QwtPicker::changed (const QwtPolygon & *pa*) [signal]

A signal emitted when the active selection has been changed. This might happen when the observed widget is resized.

Parameters

<i>pa</i>	Changed selection
-----------	-------------------

See also

[stretchSelection\(\)](#)

12.41.4.6 void QwtPicker::drawRubberBand (QPainter * *painter*) const [virtual]

Draw a rubberband , depending on [rubberBand\(\)](#) and [selectionFlags\(\)](#)

Parameters

<i>painter</i>	Painter, initialized with clip rect
----------------	-------------------------------------

See also

[rubberBand\(\)](#), [RubberBand](#), [selectionFlags\(\)](#)

12.41.4.7 void QwtPicker::drawTracker (QPainter * *painter*) const [virtual]

Draw the tracker

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[trackerRect\(\)](#), [trackerText\(\)](#)

12.41.4.8 `bool QwtPicker::end (bool ok = true) [protected, virtual]`

Close a selection setting the state to inactive.

The selection is validated and maybe fixed by [QwtPicker::accept\(\)](#).

Parameters

<i>ok</i>	If true, complete the selection and emit a selected signal otherwise discard the selection.
-----------	---

Returns

true if the selection is accepted, false otherwise

See also

[isActive\(\)](#), [begin\(\)](#), [append\(\)](#), [move\(\)](#), [selected\(\)](#), [accept\(\)](#)

Reimplemented in [QwtPlotPicker](#), and [QwtPlotZoomer](#).

12.41.4.9 `bool QwtPicker::eventFilter (QObject * o, QEvent * e) [virtual]`

Event filter.

When [isEnabled\(\)](#) == true all events of the observed widget are filtered. Mouse and keyboard events are translated into widgetMouse- and widgetKey- and widgetWheel-events. Paint and Resize events are handled to keep rubberband and tracker up to date.

See also

[event\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.41.4.10 `void QwtEventPattern::initKeyPattern () [inherited]`

Set default mouse patterns.

See also

[KeyPatternCode](#)

12.41.4.11 void QwtEventPattern::initMousePattern (int *numButtons*) [*inherited*]

Set default mouse patterns, depending on the number of mouse buttons

Parameters

<i>numButtons</i>	Number of mouse buttons (<= 3)
-------------------	----------------------------------

See also

[MousePatternCode](#)

12.41.4.12 bool QwtPicker::isActive () const

A picker is active between [begin\(\)](#) and [end\(\)](#).

Returns

true if the selection is active.

12.41.4.13 bool QwtPicker::isEnabled () const

Returns

true when enabled, false otherwise

See also

[setEnabled\(\)](#), [eventFilter\(\)](#)

12.41.4.14 bool QwtEventPattern::keyMatch (uint *pattern*, const QKeyEvent * *e*) const [*inherited*]

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.41.4.15 `bool QwtEventPattern::keyMatch (const KeyPattern & pattern,
const QKeyEvent * e) const` `[protected, virtual,
inherited]`

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Key event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.41.4.16 `const QwtArray< QwtEventPattern::KeyPattern > &
QwtEventPattern::keyPattern () const` `[inherited]`

Return key patterns.

12.41.4.17 `QwtArray< QwtEventPattern::KeyPattern > &
QwtEventPattern::keyPattern ()` `[inherited]`

Return Key patterns.

12.41.4.18 `bool QwtEventPattern::mouseMatch (const MousePattern
& pattern, const QMouseEvent * e) const` `[protected,
virtual, inherited]`

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Mouse event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.41.4.19 `bool QwtEventPattern::mouseMatch (uint pattern, const QMouseEvent * e) const` `[inherited]`

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.41.4.20 `const QwtArray< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern () const` `[inherited]`

Return mouse patterns.

12.41.4.21 `QwtArray< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern ()` `[inherited]`

Return ,ouse patterns.

12.41.4.22 `void QwtPicker::move (const QPoint & pos)` [**protected**, **virtual**]

Move the last point of the selection The [moved\(\)](#) signal is emitted.

Parameters

<i>pos</i>	New position
------------	--------------

See also

[isActive\(\)](#), [begin\(\)](#), [end\(\)](#), [append\(\)](#)

Reimplemented in [QwtPlotPicker](#).

12.41.4.23 `void QwtPicker::moved (const QPoint & pos)` [**signal**]

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

Parameters

<i>pos</i>	Position of the moved last point of the selection.
------------	--

See also

[move\(\)](#), [appended\(\)](#)

12.41.4.24 `QWidget * QwtPicker::parentWidget ()`

Return the parent widget, where the selection happens.

12.41.4.25 `const QWidget * QwtPicker::parentWidget () const`

Return the parent widget, where the selection happens.

12.41.4.26 `QRect QwtPicker::pickRect () const` [**virtual**]

Find the area of the observed widget, where selection might happen.

Returns

[QFrame::contentsRect\(\)](#) if it is a [QFrame](#), [QWidget::rect\(\)](#) otherwise.

12.41.4.27 void QwtPicker::reset () [protected, virtual]

Reset the state machine and terminate (end(false)) the selection

12.41.4.28 QwtPicker::ResizeMode QwtPicker::resizeMode () const**Returns**

Resize mode

See also

[setResizeMode\(\)](#), [SizeMode](#)

12.41.4.29 QwtPicker::RubberBand QwtPicker::rubberBand () const**Returns**

Rubberband style

See also

[setRubberBand\(\)](#), [RubberBand](#), [rubberBandPen\(\)](#)

12.41.4.30 QPen QwtPicker::rubberBandPen () const**Returns**

Rubberband pen

See also

[setRubberBandPen\(\)](#), [rubberBand\(\)](#)

12.41.4.31 const QWidget * QwtPicker::rubberBandWidget () const [protected]**Returns**

Widget displaying the rubberband

12.41.4.32 void QwtPicker::selected (const QwtPolygon & *pa*) [signal]

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

Parameters

<i>pa</i> Selected points

12.41.4.33 const QwtPolygon & QwtPicker::selection () const

Return Selected points.

12.41.4.34 int QwtPicker::selectionFlags () const**Returns**

Selection flags, an Or'd value of SelectionType, RectSelectionType and SelectionMode.

See also

[setSelectionFlags\(\)](#), [SelectionType](#), [RectSelectionType](#), [SelectionMode](#)

12.41.4.35 void QwtPicker::setEnabled (bool *enabled*) [virtual]

En/disable the picker.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>enabled</i> true or false

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.41.4.36 void QwtEventPattern::setKeyPattern (const QwtArray< KeyPattern > & *pattern*) [inherited]

Change the key event patterns.

12.41.4.37 void QwtEventPattern::setKeyPattern (uint *pattern*, int *key*, int *state* = Qt::NoButton) [inherited]

Change one key pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>key</i>	Key
<i>state</i>	State

See also

QKeyEvent

12.41.4.38 void QwtEventPattern::setMousePattern (const QwtArray< MousePattern > &*pattern*) [inherited]

Change the mouse event patterns.

12.41.4.39 void QwtEventPattern::setMousePattern (uint *pattern*, int *button*, int *state* = Qt::NoButton) [inherited]

Change one mouse pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>button</i>	Button
<i>state</i>	State

See also

QMouseEvent

12.41.4.40 void QwtPicker::setResizeMode (ResizeMode *mode*) [virtual]

Set the resize mode.

The resize mode controls what to do with the selected points of an active selection when the observed widget is resized.

Stretch means the points are scaled according to the new size, KeepSize means the points remain unchanged.

The default mode is Stretch.

Parameters

<i>mode</i>	Resize mode
-------------	-------------

See also

[resizeMode\(\)](#), [ResizeMode](#)

12.41.4.41 `void QwtPicker::setRubberBand (RubberBand rubberBand)`
[virtual]

Set the rubberband style

Parameters

<i>rubberBand</i>	Rubberband style The default value is NoRubberBand.
-------------------	---

See also

[rubberBand\(\)](#), [RubberBand](#), [setRubberBandPen\(\)](#)

12.41.4.42 `void QwtPicker::setRubberBandPen (const QPen & pen)`
[virtual]

Set the pen for the rubberband

Parameters

<i>pen</i>	Rubberband pen
------------	----------------

See also

[rubberBandPen\(\)](#), [setRubberBand\(\)](#)

12.41.4.43 `void QwtPicker::setSelectionFlags (int flags)` **[virtual]**

Set the selection flags

Parameters

<i>flags</i>	Or'd value of SelectionType, RectSelectionType and SelectionMode. The default value is NoSelection.
--------------	---

See also

[selectionFlags\(\)](#), [SelectionType](#), [RectSelectionType](#), [SelectionMode](#)

Reimplemented in [QwtPlotZoomer](#).

12.41.4.44 `void QwtPicker::setTrackerFont (const QFont & font)`
[virtual]

Set the font for the tracker

Parameters

<i>font</i>	Tracker font
-------------	--------------

See also

[trackerFont\(\)](#), [setTrackerMode\(\)](#), [setTrackerPen\(\)](#)

12.41.4.45 `void QwtPicker::setTrackerMode (DisplayMode mode)`
[virtual]

Set the display mode of the tracker.

A tracker displays information about current position of the cursor as a string. The display mode controls if the tracker has to be displayed whenever the observed widget has focus and cursor (AlwaysOn), never (AlwaysOff), or only when the selection is active (ActiveOnly).

Parameters

<i>mode</i>	Tracker display mode
-------------	----------------------

Warning

In case of AlwaysOn, mouseTracking will be enabled for the observed widget.

See also

[trackerMode\(\)](#), [DisplayMode](#)

12.41.4.46 `void QwtPicker::setTrackerPen (const QPen & pen)` [virtual]

Set the pen for the tracker

Parameters

<i>pen</i>	Tracker pen
------------	-------------

See also

[trackerPen\(\)](#), [setTrackerMode\(\)](#), [setTrackerFont\(\)](#)

12.41.4.47 QwtPickerMachine * QwtPicker::stateMachine (*int flags*) const [protected, virtual]

Create a state machine depending on the selection flags.

- PointSelection | ClickSelection
QwtPickerClickPointMachine()
- PointSelection | DragSelection
QwtPickerDragPointMachine()
- RectSelection | ClickSelection
QwtPickerClickRectMachine()
- RectSelection | DragSelection
QwtPickerDragRectMachine()
- PolygonSelection
QwtPickerPolygonMachine()

See also

[setSelectionFlags\(\)](#)

12.41.4.48 void QwtPicker::stretchSelection (const QSize & *oldSize*, const QSize & *newSize*) [protected, virtual]

Scale the selection by the ratios of *oldSize* and *newSize* The [changed\(\)](#) signal is emitted.

Parameters

<i>oldSize</i>	Previous size
<i>newSize</i>	Current size

See also

[ResizeMode](#), [setResizeMode\(\)](#), [resizeMode\(\)](#)

12.41.4.49 QFont QwtPicker::trackerFont () const

Returns

Tracker font

See also

[setTrackerFont\(\)](#), [trackerMode\(\)](#), [trackerPen\(\)](#)

12.41.4.50 QwtPicker::DisplayMode QwtPicker::trackerMode () const**Returns**

Tracker display mode

See also

[setTrackerMode\(\)](#), [DisplayMode](#)

12.41.4.51 QPen QwtPicker::trackerPen () const**Returns**

Tracker pen

See also

[setTrackerPen\(\)](#), [trackerMode\(\)](#), [trackerFont\(\)](#)

12.41.4.52 QPoint QwtPicker::trackerPosition () const**Returns**

Current position of the tracker

12.41.4.53 QRect QwtPicker::trackerRect (const QFont & *font*) const

Calculate the bounding rectangle for the tracker text from the current position of the tracker

Parameters

<i>font</i>	Font of the tracker text
-------------	--------------------------

Returns

Bounding rectangle of the tracker text

See also

[trackerPosition\(\)](#)

12.41.4.54 **QwtText QwtPicker::trackerText (const QPoint & *pos*) const** [virtual]

Return the label for a position.

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 string conversion is "%d".

Parameters

<i>pos</i>	Position
------------	----------

Returns

Converted position as string

Reimplemented in [QwtPlotPicker](#).

12.41.4.55 **const QWidget * QwtPicker::trackerWidget () const** [protected]

Returns

Widget displaying the tracker text

12.41.4.56 **void QwtPicker::transition (const QEvent * *e*)** [protected, virtual]

Passes an event to the state machine and executes the resulting commands. Append and Move commands use the current position of the cursor (QCursor::pos()).

Parameters

<i>e</i>	Event
----------	-------

12.41.4.57 **void QwtPicker::updateDisplay ()** [protected, virtual]

Update the state of rubberband and tracker label.

12.41.4.58 void QwtPicker::widgetKeyPressEvent (QKeyEvent * *ke*)
[protected, virtual]

Handle a key press event for the observed widget.

Selections can be completely done by the keyboard. The arrow keys move the cursor, the abort key aborts a selection. All other keys are handled by the current state machine.

See also

[QwtPicker](#), [selectionFlags\(\)](#)
[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#), [stateMachine\(\)](#), [QwtEventPattern::KeyPatternCode](#)

Reimplemented in [QwtPlotZoomer](#).

12.41.4.59 void QwtPicker::widgetKeyReleaseEvent (QKeyEvent * *ke*)
[protected, virtual]

Handle a key release event for the observed widget.

Passes the event to the state machine.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [stateMachine\(\)](#)

12.41.4.60 void QwtPicker::widgetLeaveEvent (QEvent *) [protected, virtual]

Handle a leave event for the observed widget.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.41.4.61 void QwtPicker::widgetMouseDoubleClickEvent (QMouseEvent * *me*) [protected, virtual]

Handle mouse double click event for the observed widget.

Empty implementation, does nothing.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

**12.41.4.62 void QwtPicker::widgetMouseMoveEvent (QMouseEvent * e)
[protected, virtual]**

Handle a mouse move event for the observed widget.

Move the last point of the selection in case of [isActive\(\)](#) == true

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

**12.41.4.63 void QwtPicker::widgetMousePressEvent (QMouseEvent * e)
[protected, virtual]**

Handle a mouse press event for the observed widget.

Begin and/or end a selection depending on the selection flags.

See also

[QwtPicker](#), [selectionFlags\(\)](#)
[eventFilter\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

**12.41.4.64 void QwtPicker::widgetMouseReleaseEvent (QMouseEvent * e)
[protected, virtual]**

Handle a mouse release event for the observed widget.

End a selection depending on the selection flags.

See also

[QwtPicker](#), [selectionFlags\(\)](#)
[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

Reimplemented in [QwtPlotZoomer](#).

12.41.4.65 void QwtPicker::widgetWheelEvent (QWheelEvent * e) [protected, virtual]

Handle a wheel event for the observed widget.

Move the last point of the selection in case of `isActive() == true`

See also

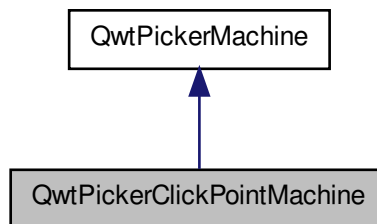
[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.42 QwtPickerClickPointMachine Class Reference

A state machine for point selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerClickPointMachine:



Public Types

- enum [Command](#) {
 Begin,
 Append,
 Move,
 End }
• typedef QList< [Command](#) > **CommandList**

Public Member Functions

- void [reset](#) ()

- void [setState](#) (int)
- int [state](#) () const
- virtual CommandList [transition](#) (const [QwtEventPattern](#) &, const QEvent *)

12.42.1 Detailed Description

A state machine for point selections. Pressing [QwtEventPattern::MouseSelect1](#) or [QwtEventPattern::KeySelect1](#) selects a point.

See also

[QwtEventPattern::MousePatternCode](#), [QwtEventPattern::KeyPatternCode](#)

12.42.2 Member Enumeration Documentation

12.42.2.1 enum QwtPickerMachine::Command [inherited]

Commands - the output of the state machine.

12.42.3 Member Function Documentation

12.42.3.1 void QwtPickerMachine::reset () [inherited]

Set the current state to 0.

12.42.3.2 void QwtPickerMachine::setState (int *state*) [inherited]

Change the current state.

12.42.3.3 int QwtPickerMachine::state () const [inherited]

Return the current state.

12.42.3.4 QwtPickerMachine::CommandList QwtPickerClickPointMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

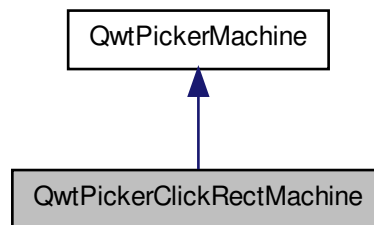
Implements [QwtPickerMachine](#).

12.43 QwtPickerClickRectMachine Class Reference

A state machine for rectangle selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerClickRectMachine:



Public Types

- enum [Command](#) {
 Begin,
 Append,
 Move,
 End }
• typedef QList< [Command](#) > **CommandList**

Public Member Functions

- void [reset](#) ()
- void [setState](#) (int)
- int [state](#) () const
- virtual CommandList [transition](#) (const [QwtEventPattern](#) &, const QEvent *)

12.43.1 Detailed Description

A state machine for rectangle selections. Pressing [QwtEventPattern::MouseSelect1](#) starts the selection, releasing it selects the first point. Pressing it again selects the second point and terminates the selection. Pressing [QwtEventPattern::KeySelect1](#) also

starts the selection, a second press selects the first point. A third one selects the second point and terminates the selection.

See also

[QwtEventPattern::MousePatternCode](#), [QwtEventPattern::KeyPatternCode](#)

12.43.2 Member Enumeration Documentation**12.43.2.1 enum QwtPickerMachine::Command [inherited]**

Commands - the output of the state machine.

12.43.3 Member Function Documentation**12.43.3.1 void QwtPickerMachine::reset () [inherited]**

Set the current state to 0.

12.43.3.2 void QwtPickerMachine::setState (int *state*) [inherited]

Change the current state.

12.43.3.3 int QwtPickerMachine::state () const [inherited]

Return the current state.

12.43.3.4 QwtPickerMachine::CommandList QwtPickerClickRectMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

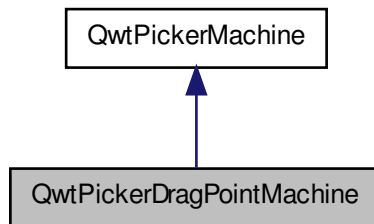
Implements [QwtPickerMachine](#).

12.44 QwtPickerDragPointMachine Class Reference

A state machine for point selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerDragPointMachine:



Public Types

- enum `Command` {
 Begin,
 Append,
 Move,
 End }
• typedef `QList< Command >` `CommandList`

Public Member Functions

- void `reset` ()
- void `setState` (int)
- int `state` () const
- virtual `CommandList transition` (const `QwtEventPattern` &, const `QEvent` *)

12.44.1 Detailed Description

A state machine for point selections. Pressing `QwtEventPattern::MouseSelect1` or `QwtEventPattern::KeySelect1` starts the selection, releasing `QwtEventPattern::MouseSelect1` or a second press of `QwtEventPattern::KeySelect1` terminates it.

12.44.2 Member Enumeration Documentation

12.44.2.1 enum `QwtPickerMachine::Command` [inherited]

Commands - the output of the state machine.

12.44.3 Member Function Documentation

12.44.3.1 void QwtPickerMachine::reset () [inherited]

Set the current state to 0.

12.44.3.2 void QwtPickerMachine::setState (int *state*) [inherited]

Change the current state.

12.44.3.3 int QwtPickerMachine::state () const [inherited]

Return the current state.

12.44.3.4 QwtPickerMachine::CommandList QwtPickerDragPointMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

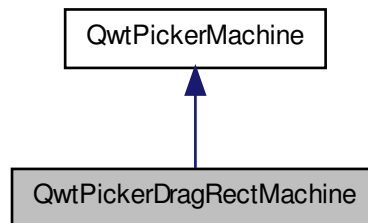
Implements [QwtPickerMachine](#).

12.45 QwtPickerDragRectMachine Class Reference

A state machine for rectangle selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerDragRectMachine:



Public Types

- enum [Command](#) {
 Begin,
 Append,
 Move,
 End }
• typedef QList< [Command](#) > **CommandList**

Public Member Functions

- void [reset](#) ()
- void [setState](#) (int)
- int [state](#) () const
- virtual CommandList [transition](#) (const [QwtEventPattern](#) &, const QEvent *)

12.45.1 Detailed Description

A state machine for rectangle selections. Pressing [QwtEventPattern::MouseSelect1](#) selects the first point, releasing it the second point. Pressing [QwtEventPattern::KeySelect1](#) also selects the first point, a second press selects the second point and terminates the selection.

See also

[QwtEventPattern::MousePatternCode](#), [QwtEventPattern::KeyPatternCode](#)

12.45.2 Member Enumeration Documentation

12.45.2.1 enum QwtPickerMachine::Command [inherited]

Commands - the output of the state machine.

12.45.3 Member Function Documentation

12.45.3.1 void QwtPickerMachine::reset () [inherited]

Set the current state to 0.

12.45.3.2 void QwtPickerMachine::setState (int *state*) [inherited]

Change the current state.

12.45.3.3 int QwtPickerMachine::state () const [inherited]

Return the current state.

12.45.3.4 QwtPickerMachine::CommandList QwtPickerDragRectMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

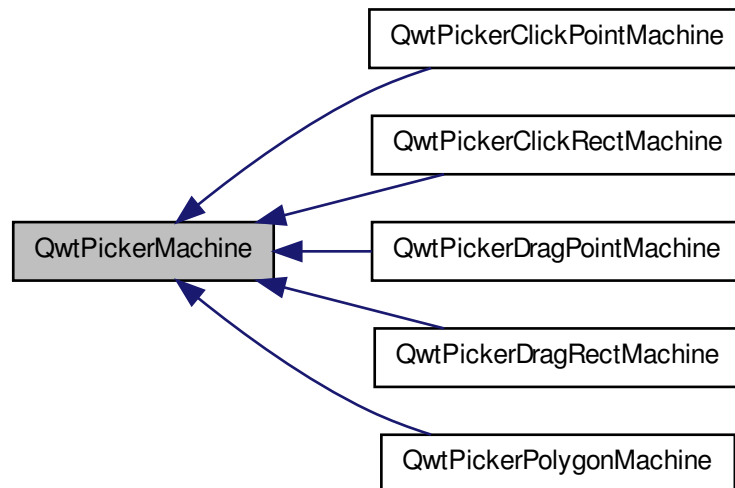
Implements [QwtPickerMachine](#).

12.46 QwtPickerMachine Class Reference

A state machine for [QwtPicker](#) selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerMachine:



Public Types

- enum [Command](#) {
 Begin,
 Append,
 Move,
 End }
• typedef QList< [Command](#) > **CommandList**

Public Member Functions

- void [reset](#) ()
- void [setState](#) (int)
- int [state](#) () const
- virtual CommandList [transition](#) (const [QwtEventPattern](#) &, const QEvent *)=0
- virtual [~QwtPickerMachine](#) ()

Protected Member Functions

- [QwtPickerMachine](#) ()

12.46.1 Detailed Description

A state machine for [QwtPicker](#) selections. [QwtPickerMachine](#) accepts key and mouse events and translates them into selection commands.

See also

[QwtEventPattern::MousePatternCode](#), [QwtEventPattern::KeyPatternCode](#)

12.46.2 Member Enumeration Documentation

12.46.2.1 enum QwtPickerMachine::Command

Commands - the output of the state machine.

12.46.3 Constructor & Destructor Documentation

12.46.3.1 QwtPickerMachine::~QwtPickerMachine () [virtual]

Destructor.

12.46.3.2 QwtPickerMachine::QwtPickerMachine () [protected]

Constructor.

12.46.4 Member Function Documentation

12.46.4.1 void QwtPickerMachine::reset ()

Set the current state to 0.

12.46.4.2 void QwtPickerMachine::setState (int *state*)

Change the current state.

12.46.4.3 int QwtPickerMachine::state () const

Return the current state.

12.46.4.4 virtual CommandList QwtPickerMachine::transition (const QwtEventPattern &, const QEvent *) [pure virtual]

Transition.

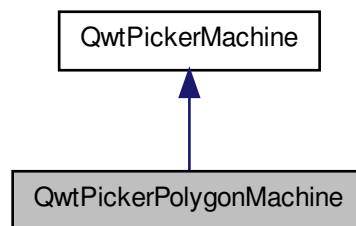
Implemented in [QwtPickerClickPointMachine](#), [QwtPickerDragPointMachine](#), [QwtPickerClickRectMachine](#), [QwtPickerDragRectMachine](#), and [QwtPickerPolygonMachine](#).

12.47 QwtPickerPolygonMachine Class Reference

A state machine for polygon selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerPolygonMachine:



Public Types

- enum [Command](#) {
 Begin,
 Append,
 Move,
 End }
• typedef QList< [Command](#) > **CommandList**

Public Member Functions

- void [reset](#) ()
- void [setState](#) (int)
- int [state](#) () const
- virtual CommandList [transition](#) (const [QwtEventPattern](#) &, const QEvent *)

12.47.1 Detailed Description

A state machine for polygon selections. Pressing `QwtEventPattern::MouseSelect1` or `QwtEventPattern::KeySelect1` starts the selection and selects the first point, or appends a point. Pressing `QwtEventPattern::MouseSelect2` or `QwtEventPattern::KeySelect2` appends the last point and terminates the selection.

See also

[QwtEventPattern::MousePatternCode](#), [QwtEventPattern::KeyPatternCode](#)

12.47.2 Member Enumeration Documentation

12.47.2.1 enum QwtPickerMachine::Command [inherited]

Commands - the output of the state machine.

12.47.3 Member Function Documentation

12.47.3.1 void QwtPickerMachine::reset () [inherited]

Set the current state to 0.

12.47.3.2 void QwtPickerMachine::setState (int *state*) [inherited]

Change the current state.

12.47.3.3 int QwtPickerMachine::state () const [inherited]

Return the current state.

12.47.3.4 QwtPickerMachine::CommandList QwtPickerPolygonMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

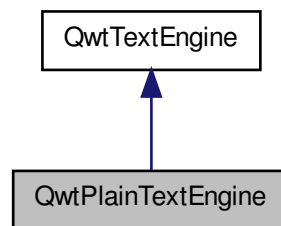
Implements [QwtPickerMachine](#).

12.48 QwtPlainTextEngine Class Reference

A text engine for plain texts.

```
#include <qwt_text_engine.h>
```

Inheritance diagram for QwtPlainTextEngine:



Public Member Functions

- virtual void [draw](#) (QPainter *painter, const QRect &rect, int flags, const QString &text) const
- virtual int [heightForWidth](#) (const QFont &font, int flags, const QString &text, int width) const
- virtual bool [mightRender](#) (const QString &) const
- [QwtPlainTextEngine](#) ()
- virtual void [textMargins](#) (const QFont &, const QString &, int &left, int &right, int &top, int &bottom) const
- virtual QSize [textSize](#) (const QFont &font, int flags, const QString &text) const
- virtual [~QwtPlainTextEngine](#) ()

12.48.1 Detailed Description

A text engine for plain texts. [QwtPlainTextEngine](#) renders texts using the basic Qt classes QPainter and QFontMetrics.

12.48.2 Constructor & Destructor Documentation

12.48.2.1 QwtPlainTextEngine::QwtPlainTextEngine ()

Constructor.

12.48.2.2 QwtPlainTextEngine::~QwtPlainTextEngine () [virtual]

Destructor.

12.48.3 Member Function Documentation**12.48.3.1 void QwtPlainTextEngine::draw (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) const [virtual]**

Draw the text in a clipping rectangle.

A wrapper for QPainter::drawText.

Parameters

<i>painter</i>	Painter
<i>rect</i>	Clipping rectangle
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered

Implements [QwtTextEngine](#).

12.48.3.2 int QwtPlainTextEngine::heightForWidth (const QFont & *font*, int *flags*, const QString & *text*, int *width*) const [virtual]

Find the height for a given width

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered
<i>width</i>	Width

Returns

Calculated height

Implements [QwtTextEngine](#).

12.48.3.3 bool QwtPlainTextEngine::mightRender (const QString &) const [virtual]

Test if a string can be rendered by this text engine.

Returns

Always true. All texts can be rendered by [QwtPlainTextEngine](#)

Implements [QwtTextEngine](#).

12.48.3.4 `void QwtPlainTextEngine::textMargins (const QFont & font, const QString & , int & left, int & right, int & top, int & bottom) const [virtual]`

Return margins around the texts

Parameters

<i>font</i>	Font of the text
<i>left</i>	Return 0
<i>right</i>	Return 0
<i>top</i>	Return value for the top margin
<i>bottom</i>	Return value for the bottom margin

Implements [QwtTextEngine](#).

12.48.3.5 `QSize QwtPlainTextEngine::textSize (const QFont & font, int flags, const QString & text) const [virtual]`

Returns the size, that is needed to render text

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered

Returns

Caluclated size

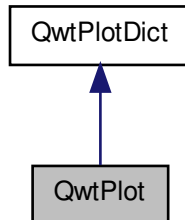
Implements [QwtTextEngine](#).

12.49 QwtPlot Class Reference

A 2-D plotting widget.

```
#include <qwt_plot.h>
```


Inheritance diagram for QwtPlot:



Public Types

- enum [Axis](#) {
 yLeft,
 yRight,
 xBottom,
 xTop,
 axisCnt }
- enum [LegendPosition](#) {
 LeftLegend,
 RightLegend,
 BottomLegend,
 TopLegend,
 ExternalLegend }

Public Slots

- void [autoRefresh](#) ()
- virtual void [clear](#) ()
- virtual void [replot](#) ()

Signals

- void [legendChecked](#) ([QwtPlotItem](#) *plotItem, bool on)
- void [legendClicked](#) ([QwtPlotItem](#) *plotItem)

Public Member Functions

- void [applyProperties](#) (const QString &)
- bool [autoDelete](#) () const
- bool [autoReplot](#) () const
- bool [axisAutoScale](#) (int axisId) const
- bool [axisEnabled](#) (int axisId) const
- QFont [axisFont](#) (int axisId) const
- int [axisMaxMajor](#) (int axisId) const
- int [axisMaxMinor](#) (int axisId) const
- const [QwtScaleDiv](#) * [axisScaleDiv](#) (int axisId) const
- [QwtScaleDiv](#) * [axisScaleDiv](#) (int axisId)
- const [QwtScaleDraw](#) * [axisScaleDraw](#) (int axisId) const
- [QwtScaleDraw](#) * [axisScaleDraw](#) (int axisId)
- [QwtScaleEngine](#) * [axisScaleEngine](#) (int axisId)
- const [QwtScaleEngine](#) * [axisScaleEngine](#) (int axisId) const
- double [axisStepSize](#) (int axisId) const
- [QwtText](#) [axisTitle](#) (int axisId) const
- const [QwtScaleWidget](#) * [axisWidget](#) (int axisId) const
- [QwtScaleWidget](#) * [axisWidget](#) (int axisId)
- [QwtPlotCanvas](#) * [canvas](#) ()
- const [QwtPlotCanvas](#) * [canvas](#) () const
- const QColor & [canvasBackground](#) () const
- int [canvasLineWidth](#) () const
- virtual [QwtScaleMap](#) [canvasMap](#) (int axisId) const
- void [detachItems](#) (int rtti=QwtPlotItem::Rtti_PlotItem, bool autoDelete=true)
- virtual void [drawCanvas](#) (QPainter *)
- void [enableAxis](#) (int axisId, bool tf=true)
- virtual bool [event](#) (QEvent *)
- QString [grabProperties](#) () const
- void [insertLegend](#) ([QwtLegend](#) *, [LegendPosition](#)=QwtPlot::RightLegend, double ratio=-1.0)
- double [invTransform](#) (int axisId, int pos) const
- const [QwtPlotItemList](#) & [itemList](#) () const
- [QwtLegend](#) * [legend](#) ()
- const [QwtLegend](#) * [legend](#) () const
- int [margin](#) () const
- virtual QSize [minimumSizeHint](#) () const
- [QwtPlotLayout](#) * [plotLayout](#) ()
- const [QwtPlotLayout](#) * [plotLayout](#) () const
- virtual void [polish](#) ()
- void [print](#) (QPaintDevice &p, const [QwtPlotPrintFilter](#) &=QwtPlotPrintFilter()) const
- virtual void [print](#) (QPainter *, const QRect &rect, const [QwtPlotPrintFilter](#) &=QwtPlotPrintFilter()) const
- [QwtPlot](#) (const [QwtText](#) &title, QWidget *p=NULL)
- [QwtPlot](#) (QWidget *p=NULL)

- void [setAutoDelete](#) (bool)
- void [setAutoReplot](#) (bool tf=true)
- void [setAxisAutoScale](#) (int axisId)
- void [setAxisFont](#) (int axisId, const QFont &f)
- void [setAxisLabelAlignment](#) (int axisId, Qt::Alignment)
- void [setAxisLabelRotation](#) (int axisId, double rotation)
- void [setAxisMaxMajor](#) (int axisId, int maxMajor)
- void [setAxisMaxMinor](#) (int axisId, int maxMinor)
- void [setAxisScale](#) (int axisId, double min, double max, double step=0)
- void [setAxisScaleDiv](#) (int axisId, const [QwtScaleDiv](#) &)
- void [setAxisScaleDraw](#) (int axisId, [QwtScaleDraw](#) *)
- void [setAxisScaleEngine](#) (int axisId, [QwtScaleEngine](#) *)
- void [setAxisTitle](#) (int axisId, const [QwtText](#) &)
- void [setAxisTitle](#) (int axisId, const QString &)
- void [setCanvasBackground](#) (const QColor &c)
- void [setCanvasLineWidth](#) (int w)
- void [setMargin](#) (int margin)
- void [setTitle](#) (const [QwtText](#) &t)
- void [setTitle](#) (const QString &)
- virtual QSize [sizeHint](#) () const
- [QwtText](#) [title](#) () const
- [QwtTextLabel](#) * [titleLabel](#) ()
- const [QwtTextLabel](#) * [titleLabel](#) () const
- int [transform](#) (int axisId, double value) const
- void [updateAxes](#) ()
- virtual void [updateLayout](#) ()
- virtual [~QwtPlot](#) ()

Protected Slots

- virtual void [legendItemChecked](#) (bool)
- virtual void [legendItemClicked](#) ()

Protected Member Functions

- virtual void [drawItems](#) (QPainter *, const QRect &, const [QwtScaleMap](#) maps[axisCnt], const [QwtPlotPrintFilter](#) &) const
- virtual void [printCanvas](#) (QPainter *, const QRect &boundingRect, const QRect &canvasRect, const [QwtScaleMap](#) maps[axisCnt], const [QwtPlotPrintFilter](#) &) const
- virtual void [printLegend](#) (QPainter *, const QRect &) const
- virtual void [printLegendItem](#) (QPainter *, const QWidget *, const QRect &) const
- virtual void [printScale](#) (QPainter *, int axisId, int startDist, int endDist, int baseDist, const QRect &) const
- virtual void [printTitle](#) (QPainter *, const QRect &) const
- virtual void [resizeEvent](#) (QResizeEvent *e)
- virtual void [updateTabOrder](#) ()

Static Protected Member Functions

- static bool [axisValid](#) (int axisId)

12.49.1 Detailed Description

A 2-D plotting widget. [QwtPlot](#) is a widget for plotting two-dimensional graphs. An unlimited number of plot items can be displayed on its canvas. Plot items might be curves ([QwtPlotCurve](#)), markers ([QwtPlotMarker](#)), the grid ([QwtPlotGrid](#)), or anything else derived from [QwtPlotItem](#). A plot can have up to four axes, with each plot item attached to an x- and a y axis. The scales at the axes can be explicitly set ([QwtScaleDiv](#)), or are calculated from the plot items, using algorithms ([QwtScaleEngine](#)) which can be configured separately for each axis.

Example

The following example shows (schematically) the most simple way to use [QwtPlot](#). By default, only the left and bottom axes are visible and their scales are computed automatically.

```
#include <qwt_plot.h>
#include <qwt_plot_curve.h>

QwtPlot *myPlot = new QwtPlot("Two Curves", parent);

// add curves
QwtPlotCurve *curve1 = new QwtPlotCurve("Curve 1");
QwtPlotCurve *curve2 = new QwtPlotCurve("Curve 2");

// copy the data into the curves
curve1->setData(...);
curve2->setData(...);

curve1->attach(myPlot);
curve2->attach(myPlot);

// finally, refresh the plot
myPlot->replot();
```

12.49.2 Member Enumeration Documentation

12.49.2.1 enum QwtPlot::Axis

Axis index

- yLeft
- yRight
- xBottom
- xTop

12.49.2.2 enum QwtPlot::LegendPosition

Position of the legend, relative to the canvas.

- LeftLegend
The legend will be left from the yLeft axis.
- RightLegend
The legend will be right from the yLeft axis.
- BottomLegend
The legend will be right below the xBottom axis.
- TopLegend
The legend will be between xTop axis and the 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

In case of ExternalLegend, the legend is not printed by [print\(\)](#).

See also

[insertLegend\(\)](#)

12.49.3 Constructor & Destructor Documentation

12.49.3.1 QwtPlot::QwtPlot (QWidget * *parent* = *NULL*) [explicit]

Constructor.

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.49.3.2 QwtPlot::QwtPlot (const QwtText & *title*, QWidget * *parent* = *NULL*) [explicit]

Constructor.

Parameters

<i>title</i>	Title text
<i>parent</i>	Parent widget

12.49.3.3 QwtPlot::~~QwtPlot () [virtual]

Destructor.

12.49.4 Member Function Documentation**12.49.4.1 void QwtPlot::applyProperties (const QString &)**

This method is intended for manipulating the plot widget from a specific editor in the Qwt designer plugin.

Warning

The plot editor has never been implemented.

12.49.4.2 bool QwtPlotDict::autoDelete () const [inherited]**Returns**

true if auto deletion is enabled

See also

[setAutoDelete\(\)](#), [attachItem\(\)](#)

12.49.4.3 void QwtPlot::autoRefresh () [slot]

Replots the plot if [QwtPlot::autoReplot\(\)](#) is true.

12.49.4.4 bool QwtPlot::autoReplot () const**Returns**

true if the autoReplot option is set.

12.49.4.5 bool QwtPlot::axisAutoScale (int *axisId*) const**Returns**

true if autoscaling is enabled

Parameters

<i>axisId</i> axis index

12.49.4.6 bool QwtPlot::axisEnabled (int *axisId*) const**Returns**

true if a specified axis is enabled

Parameters

<i>axisId</i> axis index

12.49.4.7 QFont QwtPlot::axisFont (int *axisId*) const**Returns**

the font of the scale labels for a specified axis

Parameters

<i>axisId</i> axis index

12.49.4.8 int QwtPlot::axisMaxMajor (int *axisId*) const**Returns**

the maximum number of major ticks for a specified axis

Parameters

<i>axisId</i> axis index sa setAxisMaxMajor()

12.49.4.9 int QwtPlot::axisMaxMinor (int *axisId*) const**Returns**

the maximum number of minor ticks for a specified axis

Parameters

<i>axisId</i>	axis index sa setAxisMaxMinor()
---------------	---

12.49.4.10 const QwtScaleDiv * QwtPlot::axisScaleDiv (int *axisId*) const

Return the scale division of a specified axis.

`axisScaleDiv(axisId)->lowerBound()`, `axisScaleDiv(axisId)->upperBound()` are the current limits of the axis scale.

Parameters

<i>axisId</i>	axis index
---------------	------------

Returns

Scale division

See also

[QwtScaleDiv](#), [setAxisScaleDiv\(\)](#)

12.49.4.11 QwtScaleDiv * QwtPlot::axisScaleDiv (int *axisId*)

Return the scale division of a specified axis.

`axisScaleDiv(axisId)->lowerBound()`, `axisScaleDiv(axisId)->upperBound()` are the current limits of the axis scale.

Parameters

<i>axisId</i>	axis index
---------------	------------

Returns

Scale division

See also

[QwtScaleDiv](#), [setAxisScaleDiv\(\)](#)

12.49.4.12 `const QwtScaleDraw * QwtPlot::axisScaleDraw (int axisId) const`**Returns**

the scale draw of a specified axis

Parameters

<i>axisId</i>	axis index
---------------	------------

Returns

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

See also

[QwtScaleDraw](#)

12.49.4.13 `QwtScaleDraw * QwtPlot::axisScaleDraw (int axisId)`**Returns**

the scale draw of a specified axis

Parameters

<i>axisId</i>	axis index
---------------	------------

Returns

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

See also

[QwtScaleDraw](#)

12.49.4.14 `const QwtScaleEngine * QwtPlot::axisScaleEngine (int axisId) const`**Parameters**

<i>axisId</i>	axis index
---------------	------------

Returns

Scale engine for a specific axis

12.49.4.15 QwtScaleEngine * QwtPlot::axisScaleEngine (int *axisId*)**Parameters**

<i>axisId</i> axis index

Returns

Scale engine for a specific axis

12.49.4.16 double QwtPlot::axisStepSize (int *axisId*) const

Return the step size parameter, that has been set in setAxisScale. This doesn't need to be the step size of the current scale.

Parameters

<i>axisId</i> axis index

Returns

step size parameter value

See also

[setAxisScale\(\)](#)

12.49.4.17 QwtText QwtPlot::axisTitle (int *axisId*) const**Returns**

the title of a specified axis

Parameters

<i>axisId</i> axis index

12.49.4.18 bool QwtPlot::axisValid (int *axisId*) [static, protected]**Returns**

true if the specified axis exists, otherwise false

Parameters

<i>axisId</i> axis index

12.49.4.19 `const QwtScaleWidget * QwtPlot::axisWidget (int axisId) const`

Returns

specified axis, or NULL if *axisId* is invalid.

Parameters

<i>axisId</i> axis index

12.49.4.20 `QwtScaleWidget * QwtPlot::axisWidget (int axisId)`

Returns

specified axis, or NULL if *axisId* is invalid.

Parameters

<i>axisId</i> axis index

12.49.4.21 `QwtPlotCanvas * QwtPlot::canvas ()`

Returns

the plot's canvas

12.49.4.22 `const QwtPlotCanvas * QwtPlot::canvas () const`

Returns

the plot's canvas

12.49.4.23 `const QColor & QwtPlot::canvasBackground () const`

Nothing else than: `canvas()->palette().color(QPalette::Normal, QColorGroup::Background);`

Returns

the background color of the plotting area.

12.49.4.24 int QwtPlot::canvasLineWidth () const

Nothing else than: [canvas\(\)](#)->linewidth(), left for compatibility only.

Returns

the border width of the plotting area

12.49.4.25 QwtScaleMap QwtPlot::canvasMap (int *axisId*) const [virtual]**Parameters**

<i>axisId</i>	Axis
---------------	------

Returns

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

See also

[QwtScaleMap](#), [transform\(\)](#), [invTransform\(\)](#)

12.49.4.26 void QwtPlot::clear () [virtual, slot]

Remove all curves and markers

Deprecated

Use [QwtPlotDeict::detachItems](#) instead

12.49.4.27 void QwtPlotDict::detachItems (int *rtti* = *QwtPlotItem::Rtti_PlotItem*, bool *autoDelete* = true) [inherited]

Detach items from the dictionary

Parameters

<i>rtti</i>	In case of QwtPlotItem::Rtti_PlotItem detach all items otherwise only those items of the type <i>rtti</i> .
<i>autoDelete</i>	If true, delete all detached items

12.49.4.28 void QwtPlot::drawCanvas (QPainter * *painter*) [virtual]

Redraw the canvas.

Parameters

<i>painter</i>	Painter used for drawing
----------------	--------------------------

Warning

drawCanvas calls drawItems what is also used for printing. Applications that like to add individual plot items better overload [drawItems\(\)](#)

See also

[drawItems\(\)](#)

12.49.4.29 void QwtPlot::drawItems (QPainter * *painter*, const QRect & *rect*, const QwtScaleMap *map*[*axisCnt*], const QwtPlotPrintFilter & *pfilter*) const [protected, virtual]

Redraw the canvas items.

Parameters

<i>painter</i>	Painter used for drawing
<i>rect</i>	Bounding rectangle where to paint
<i>map</i>	QwtPlot::axisCnt maps, mapping between plot and paint device coordinates
<i>pfilter</i>	Plot print filter

12.49.4.30 void QwtPlot::enableAxis (int *axisId*, bool *tf* = *true*)

Enable or disable a specified axis.

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

Only xBottom and yLeft are enabled by default.

Parameters

<i>axisId</i>	axis index
<i>tf</i>	true (enabled) or false (disabled)

12.49.4.31 bool QwtPlot::event (QEvent * *e*) [virtual]

Adds handling of layout requests.

12.49.4.32 QString QwtPlot::grabProperties () const

This method is intended for manipulating the plot widget from a specific editor in the Qwt designer plugin.

Warning

The plot editor has never been implemented.

12.49.4.33 void QwtPlot::insertLegend (QwtLegend * *legend*, QwtPlot::LegendPosition *pos* = *QwtPlot::RightLegend*, double *ratio* = -1.0)

Insert a legend.

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

If *pos* != *QwtPlot::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

<i>legend</i>	Legend
<i>pos</i>	The legend's position. For top/left position the number of columns will be limited to 1, otherwise it will be set to unlimited.
<i>ratio</i>	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\(\)](#), [QwtPlotLayout::legendPosition\(\)](#), [QwtPlotLayout::setLegendPosition\(\)](#)

12.49.4.34 double QwtPlot::invTransform (int *axisId*, int *pos*) const

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

Parameters

<i>axisId</i>	axis index
<i>pos</i>	position

Warning

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

12.49.4.35 `const QwtPlotItemList & QwtPlotDict::itemList () const`
[inherited]

A QwtPlotItemList of all attached plot items.

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

Returns

List of all attached plot items.

12.49.4.36 `QwtLegend * QwtPlot::legend ()`**Returns**

the plot's legend

See also

[insertLegend\(\)](#)

12.49.4.37 `const QwtLegend * QwtPlot::legend () const`**Returns**

the plot's legend

See also

[insertLegend\(\)](#)

12.49.4.38 `void QwtPlot::legendChecked (QwtPlotItem * plotItem, bool on)`
[signal]

A signal which is emitted when the user has clicked on a legend item, which is in QwtLegend::CheckableItem mode

Parameters

<i>plotItem</i>	Corresponding plot item of the selected legend item
<i>on</i>	True when the legen item is checked

Note

clicks are disabled as default

See also

[QwtLegend::setItemMode\(\)](#), [QwtLegend::itemMode\(\)](#)

12.49.4.39 void QwtPlot::legendClicked (QwtPlotItem * *plotItem*) [**signal**]

A signal which is emitted when the user has clicked on a legend item, which is in QwtLegend::ClickableItem mode.

Parameters

<i>plotItem</i>	Corresponding plot item of the selected legend item
-----------------	---

Note

clicks are disabled as default

See also

[QwtLegend::setItemMode\(\)](#), [QwtLegend::itemMode\(\)](#)

12.49.4.40 void QwtPlot::legendItemChecked (bool *on*) [protected, virtual, slot]

Called internally when the legend has been checked Emits a [legendClicked\(\)](#) signal.

12.49.4.41 void QwtPlot::legendItemClicked () [protected, virtual, slot]

Called internally when the legend has been clicked on. Emits a [legendClicked\(\)](#) signal.

12.49.4.42 int QwtPlot::margin () const

Returns

margin

See also

[setMargin\(\)](#), [QwtPlotLayout::margin\(\)](#), [plotLayout\(\)](#)

12.49.4.43 QSize QwtPlot::minimumSizeHint () const [virtual]

Return a minimum size hint.

12.49.4.44 const QwtPlotLayout * QwtPlot::plotLayout () const**Returns**

the plot's titel label.

12.49.4.45 QwtPlotLayout * QwtPlot::plotLayout ()**Returns**

the plot's title

12.49.4.46 void QwtPlot::polish () [virtual]

Polish.

12.49.4.47 void QwtPlot::print (QPaintDevice & *paintDev*, const QwtPlotPrintFilter & *pfilter* = QwtPlotPrintFilter ()) const

Print the plot to a QPaintDevice (QPrinter) This function prints the contents of a [QwtPlot](#) instance to QPaintDevice object. The size is derived from its device metrics.

Parameters

<i>paintDev</i>	device to paint on, often a printer
<i>pfilter</i>	print filter

See also

[QwtPlotPrintFilter](#)

12.49.4.48 void QwtPlot::print (QPainter * *painter*, const QRect & *plotRect*, const QwtPlotPrintFilter & *pfilter* = QwtPlotPrintFilter ()) const [virtual]

Paint the plot into a given rectangle. Paint the contents of a [QwtPlot](#) instance into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>plotRect</i>	Bounding rectangle
<i>pfilter</i>	Print filter

See also

[QwtPlotPrintFilter](#)

12.49.4.49 void QwtPlot::printCanvas (QPainter * *painter*, const QRect & *boundingRect*, const QRect & *canvasRect*, const QwtScaleMap map[*axisCnt*], const QwtPlotPrintFilter & *pfilter*) const [protected, virtual]

Print the canvas into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>map</i>	Maps mapping between plot and paint device coordinates
<i>boundingRect</i>	Bounding rectangle
<i>canvasRect</i>	Canvas rectangle
<i>pfilter</i>	Print filter

See also

[QwtPlotPrintFilter](#)

12.49.4.50 void QwtPlot::printLegend (QPainter * *painter*, const QRect & *rect*) const [protected, virtual]

Print the legend into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>rect</i>	Bounding rectangle

12.49.4.51 void QwtPlot::printLegendItem (QPainter * *painter*, const QWidget * *w*, const QRect & *rect*) const [protected, virtual]

Print the legend item into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>w</i>	Widget representing a legend item
<i>rect</i>	Bounding rectangle

12.49.4.52 void QwtPlot::printScale (QPainter * *painter*, int *axisId*, int *startDist*, int *endDist*, int *baseDist*, const QRect & *rect*) const [protected, virtual]

Paint a scale into a given rectangle. Paint the scale into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>axisId</i>	Axis
<i>startDist</i>	Start border distance
<i>endDist</i>	End border distance
<i>baseDist</i>	Base distance
<i>rect</i>	Bounding rectangle

12.49.4.53 void QwtPlot::printTitle (QPainter * *painter*, const QRect & *rect*) const [protected, virtual]

Print the title into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>rect</i>	Bounding rectangle

12.49.4.54 void QwtPlot::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

12.49.4.55 `void QwtPlot::resizeEvent (QResizeEvent * e)` [**protected**, **virtual**]

Resize and update internal layout

Parameters

<i>e</i>	Resize event
----------	--------------

12.49.4.56 `void QwtPlotDict::setAutoDelete (bool autoDelete)` [**inherited**]

En/Disable Auto deletion

If Auto deletion is on all attached plot items will be deleted in the destructor of [QwtPlotDict](#). The default value is on.

See also[autoDelete\(\)](#), [attachItem\(\)](#)

12.49.4.57 `void QwtPlot::setAutoReplot (bool tf = true)`

Set or reset the autoReplot option.

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

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

Parameters

<i>tf</i>	true or false. Defaults to true.
-----------	----------------------------------

See also[replot\(\)](#)

12.49.4.58 void QwtPlot::setAxisAutoScale (int *axisId*)

Enable autoscaling for a specified axis.

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

Parameters

<i>axisId</i>	axis index
---------------	------------

See also

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

12.49.4.59 void QwtPlot::setAxisFont (int *axisId*, const QFont &*f*)

Change the font of an axis.

Parameters

<i>axisId</i>	axis index
<i>f</i>	font

Warning

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

12.49.4.60 void QwtPlot::setAxisLabelAlignment (int *axisId*, Qt::Alignment *alignment*)

Change the alignment of the tick labels

Parameters

<i>axisId</i>	axis index
<i>alignment</i>	Or'd Qt::AlignmentFlags <see qnamespace.h>

See also

[QwtScaleDraw::setLabelAlignment\(\)](#)

12.49.4.61 void QwtPlot::setAxisLabelRotation (int *axisId*, double *rotation*)

Rotate all tick labels

Parameters

<i>axisId</i>	axis index
<i>rotation</i>	Angle in degrees. When changing the label rotation, the label alignment might be adjusted too.

See also

[QwtScaleDraw::setLabelRotation\(\)](#), [setAxisLabelAlignment\(\)](#)

12.49.4.62 void QwtPlot::setAxisMaxMajor (int *axisId*, int *maxMajor*)

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

Parameters

<i>axisId</i>	axis index
<i>maxMajor</i>	maximum number of major steps

See also

[axisMaxMajor\(\)](#)

12.49.4.63 void QwtPlot::setAxisMaxMinor (int *axisId*, int *maxMinor*)

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

Parameters

<i>axisId</i>	axis index
<i>maxMinor</i>	maximum number of minor steps

See also

[axisMaxMinor\(\)](#)

12.49.4.64 void QwtPlot::setAxisScale (int *axisId*, double *min*, double *max*, double *stepSize* = 0)

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

Parameters

<i>axisId</i>	axis index
<i>min</i>	
<i>max</i>	minimum and maximum of the scale
<i>stepSize</i>	Major step size. If <code>step == 0</code> , the step size is calculated automatically using the <code>maxMajor</code> setting.

See also

[setAxisMaxMajor\(\)](#), [setAxisAutoScale\(\)](#)

12.49.4.65 void QwtPlot::setAxisScaleDiv (int *axisId*, const QwtScaleDiv & *scaleDiv*)

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

Parameters

<i>axisId</i>	axis index
<i>scaleDiv</i>	Scale division

See also

[setAxisScale\(\)](#), [setAxisAutoScale\(\)](#)

12.49.4.66 void QwtPlot::setAxisScaleDraw (int *axisId*, QwtScaleDraw * *scaleDraw*)

Set a scale draw.

Parameters

<i>axisId</i>	axis index
<i>scaleDraw</i>	object responsible for drawing scales.

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

See also

[QwtScaleDraw](#), [QwtScaleWidget](#)

Warning

The attributes of *scaleDraw* will be overwritten by those of the previous [QwtScaleDraw](#).

12.49.4.67 void QwtPlot::setAxisScaleEngine (int *axisId*, QwtScaleEngine * *scaleEngine*)

Change the scale engine for an axis

Parameters

<i>axisId</i>	axis index
<i>scaleEngine</i>	Scale engine

See also

[axisScaleEngine\(\)](#)

12.49.4.68 void QwtPlot::setAxisTitle (int *axisId*, const QString & *title*)

Change the title of a specified axis.

Parameters

<i>axisId</i>	axis index
<i>title</i>	axis title

12.49.4.69 void QwtPlot::setAxisTitle (int *axisId*, const QwtText & *title*)

Change the title of a specified axis.

Parameters

<i>axisId</i>	axis index
<i>title</i>	axis title

12.49.4.70 void QwtPlot::setCanvasBackground (const QColor & *c*)

Change the background of the plotting area.

Sets *c* to QColorGroup::Background of all colorgroups of the palette of the canvas. Using [canvas\(\)](#)->setPalette() is a more powerful way to set these colors.

Parameters

<i>c</i>	new background color
----------	----------------------

12.49.4.71 void QwtPlot::setCanvasLineWidth (int *w*)

Change the border width of the plotting area Nothing else than [canvas\(\)](#)->setLineWidth(*w*),

left for compatibility only.

Parameters

<i>w</i>	new border width
----------	------------------

12.49.4.72 void QwtPlot::setMargin (int *margin*)

Change the margin of the plot. The margin is the space around all components.

Parameters

<i>margin</i>	new margin
---------------	------------

See also

[QwtPlotLayout::setMargin\(\)](#), [margin\(\)](#), [plotLayout\(\)](#)

12.49.4.73 void QwtPlot::setTitle (const QwtText & *title*)

Change the plot's title

Parameters

<i>title</i>	New title
--------------	-----------

12.49.4.74 void QwtPlot::setTitle (const QString & *title*)

Change the plot's title

Parameters

<i>title</i>	New title
--------------	-----------

12.49.4.75 QSize QwtPlot::sizeHint () const [virtual]

Return sizeHint

See also

[minimumSizeHint\(\)](#)

12.49.4.76 QwtText QwtPlot::title () const

Returns

the plot's title

12.49.4.77 `const QwtTextLabel * QwtPlot::titleLabel () const`

Returns

the plot's titel label.

12.49.4.78 `QwtTextLabel * QwtPlot::titleLabel ()`

Returns

the plot's titel label.

12.49.4.79 `int QwtPlot::transform (int axisId, double value) const`

Transform a value into a coordinate in the plotting region.

Parameters

<i>axisId</i>	axis index
<i>value</i>	value

Returns

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

12.49.4.80 `void QwtPlot::updateAxes ()`

Rebuild the scales.

12.49.4.81 `void QwtPlot::updateLayout () [virtual]`

Adjust plot content to its current size.

See also

[resizeEvent\(\)](#)

12.49.4.82 void QwtPlot::updateTabOrder () [protected, virtual]

Update the focus tab order

The order is changed so that the canvas will be in front of the first legend item, or behind the last legend item - depending on the position of the legend.

12.50 QwtPlotCanvas Class Reference

Canvas of a [QwtPlot](#).

```
#include <qwt_plot_canvas.h>
```

Public Types

- enum [FocusIndicator](#) {
 NoFocusIndicator,
 CanvasFocusIndicator,
 ItemFocusIndicator }
- enum [PaintAttribute](#) {
 PaintCached = 1,
 PaintPacked = 2 }

Public Member Functions

- [FocusIndicator](#) [focusIndicator](#) () const
- void [invalidatePaintCache](#) ()
- QPixmap * [paintCache](#) ()
- const QPixmap * [paintCache](#) () const
- const [QwtPlot](#) * [plot](#) () const
- [QwtPlot](#) * [plot](#) ()
- [QwtPlotCanvas](#) ([QwtPlot](#) *)
- void [replot](#) ()
- void [setFocusIndicator](#) ([FocusIndicator](#))
- void [setPaintAttribute](#) ([PaintAttribute](#), bool on=true)
- bool [testPaintAttribute](#) ([PaintAttribute](#)) const
- virtual [~QwtPlotCanvas](#) ()

Protected Member Functions

- void [drawCanvas](#) (QPainter *painter=NULL)
- virtual void [drawContents](#) (QPainter *)
- virtual void [drawFocusIndicator](#) (QPainter *)
- virtual void [hideEvent](#) (QHideEvent *)
- virtual void [paintEvent](#) (QPaintEvent *)

12.50.1 Detailed Description

Canvas of a [QwtPlot](#).

See also

[QwtPlot](#)

12.50.2 Member Enumeration Documentation

12.50.2.1 enum QwtPlotCanvas::FocusIndicator

Focus indicator.

- NoFocusIndicator
Don't paint a focus indicator
- CanvasFocusIndicator
The focus is related to the complete canvas. Paint the focus indicator using `paintFocus()`
- ItemFocusIndicator
The focus is related to an item (curve, point, ...) on the canvas. It is up to the application to display a focus indication using f.e. highlighting.

See also

[setFocusIndicator\(\)](#), [focusIndicator\(\)](#), [paintFocus\(\)](#)

12.50.2.2 enum QwtPlotCanvas::PaintAttribute

Paint attributes.

- PaintCached

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. Disabling the cache will improve the performance for incremental paints (using [QwtPlotCurve::draw](#)).

- PaintPacked

Suppress system background repaints and paint it together with the canvas contents. Painting packed might avoid flickering for expensive repaints, when there is a notable gap between painting the background and the plot contents.

The default setting enables PaintCached and PaintPacked

See also

[setPaintAttribute\(\)](#), [testPaintAttribute\(\)](#), [paintCache\(\)](#)

12.50.3 Constructor & Destructor Documentation

12.50.3.1 QwtPlotCanvas::QwtPlotCanvas (QwtPlot * *plot*) [explicit]

Sets a cross cursor, enables QwtPlotCanvas::PaintCached.

12.50.3.2 QwtPlotCanvas::~QwtPlotCanvas () [virtual]

Destructor.

12.50.4 Member Function Documentation

12.50.4.1 void QwtPlotCanvas::drawCanvas (QPainter * *painter* = *NULL*) [protected]

Draw the the canvas

Paints all plot items to the contentsRect(), using [QwtPlot::drawCanvas](#) and updates the paint cache.

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[QwtPlot::drawCanvas\(\)](#), [setPaintAttributes\(\)](#), [testPaintAttributes\(\)](#)

12.50.4.2 void QwtPlotCanvas::drawContents (QPainter * *painter*)
[protected, virtual]

Redraw the canvas, and focus rect

Parameters

<i>painter</i>	Painter
----------------	---------

12.50.4.3 void QwtPlotCanvas::drawFocusIndicator (QPainter * *painter*)
[protected, virtual]

Draw the focus indication

Parameters

<i>painter</i>	Painter
----------------	---------

12.50.4.4 QwtPlotCanvas::FocusIndicator QwtPlotCanvas::focusIndicator ()
const

Returns

Focus indicator

See also

[FocusIndicator](#), [setFocusIndicator\(\)](#)

12.50.4.5 void QwtPlotCanvas::hideEvent (QHideEvent * *event*)
[protected, virtual]

Hide event

Parameters

<i>event</i>	Hide event
--------------	------------

12.50.4.6 void QwtPlotCanvas::invalidatePaintCache ()

Invalidate the internal paint cache.

12.50.4.7 `const QPixmap * QwtPlotCanvas::paintCache () const`

Return the paint cache, might be null.

12.50.4.8 `QPixmap * QwtPlotCanvas::paintCache ()`

Return the paint cache, might be null.

12.50.4.9 `void QwtPlotCanvas::paintEvent (QPaintEvent * event)`
[protected, virtual]

Paint event

Parameters

<i>event</i>	Paint event
--------------	-------------

12.50.4.10 `const QwtPlot * QwtPlotCanvas::plot () const`

Return parent plot widget.

12.50.4.11 `QwtPlot * QwtPlotCanvas::plot ()`

Return parent plot widget.

12.50.4.12 `void QwtPlotCanvas::replot ()`

Invalidate the paint cache and repaint the canvas

See also

[invalidatePaintCache\(\)](#)

12.50.4.13 `void QwtPlotCanvas::setFocusIndicator (FocusIndicator focusIndicator)`

Set the focus indicator

See also

[FocusIndicator](#), [focusIndicator\(\)](#)

12.50.4.14 `void QwtPlotCanvas::setPaintAttribute (PaintAttribute attribute,
bool on = true)`

Changing the paint attributes.

Parameters

<i>attribute</i>	Paint attribute
<i>on</i>	On/Off

The default setting enables PaintCached and PaintPacked

See also

[testPaintAttribute\(\)](#), [drawCanvas\(\)](#), [drawContents\(\)](#), [paintCache\(\)](#)

12.50.4.15 `bool QwtPlotCanvas::testPaintAttribute (PaintAttribute attribute)
const`

Test wether a paint attribute is enabled

Parameters

<i>attribute</i>	Paint attribute
------------------	-----------------

Returns

true if the attribute is enabled

See also

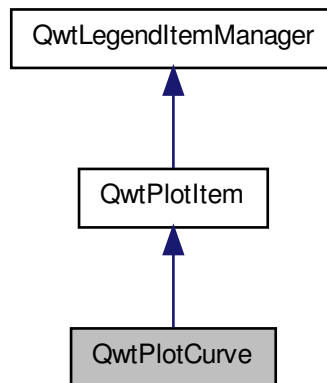
[setPaintAttribute\(\)](#)

12.51 QwtPlotCurve Class Reference

A plot item, that represents a series of points.

```
#include <qwt_plot_curve.h>
```


Inheritance diagram for QwtPlotCurve:



Public Types

- enum [CurveAttribute](#) {
 Inverted = 1,
 Fitted = 2 }
- enum [CurveStyle](#) {
 NoCurve,
 Lines,
 Sticks,
 Steps,
 Dots,
 UserCurve = 100 }
- enum [CurveType](#) {
 Yfx,
 Xfy }
- enum [ItemAttribute](#) {
 Legend = 1,
 AutoScale = 2 }
- enum [PaintAttribute](#) {
 PaintFiltered = 1,
 ClipPolygons = 2 }

- enum [RenderHint](#) { **RenderAntialiased** = 1 }
- enum [RttiValues](#) {
Rtti_PlotItem = 0,
Rtti_PlotGrid,
Rtti_PlotScale,
Rtti_PlotMarker,
Rtti_PlotCurve,
Rtti_PlotHistogram,
Rtti_PlotSpectrogram,
Rtti_PlotSVG,
Rtti_PlotUserItem = 1000 }

Public Member Functions

- void [attach](#) ([QwtPlot](#) *plot)
- double [baseline](#) () const
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- const [QBrush](#) & [brush](#) () const
- int [closestPoint](#) (const [QPoint](#) &pos, double *dist=NULL) const
- [QwtCurveFitter](#) * [curveFitter](#) () const
- [CurveType](#) [curveType](#) () const
- const [QwtData](#) & [data](#) () const
- [QwtData](#) & [data](#) ()
- int [dataSize](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &) const
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- void [draw](#) (int from, int to) const
- void [hide](#) ()
- [QwtDoubleRect](#) [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QRect](#) &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual [QWidget](#) * [legendItem](#) () const
- double [maxXValue](#) () const
- double [maxYValue](#) () const
- double [minXValue](#) () const
- double [minYValue](#) () const
- [QRect](#) [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- const [QPen](#) & [pen](#) () const
- [QwtPlot](#) * [plot](#) () const
- [QwtPlotCurve](#) (const [QwtText](#) &title)

- [QwtPlotCurve](#) ()
- [QwtPlotCurve](#) (const QString &title)
- virtual int [rtti](#) () const
- QwtDoubleRect [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAxis](#) (int xAxis, int yAxis)
- void [setBaseline](#) (double ref)
- void [setBrush](#) (const QBrush &)
- void [setCurveAttribute](#) ([CurveAttribute](#), bool on=true)
- void [setCurveFitter](#) ([QwtCurveFitter](#) *)
- void [setCurveType](#) ([CurveType](#))
- void [setData](#) (const double *xData, const double *yData, int size)
- void [setData](#) (const QwtArray< double > &xData, const QwtArray< double > &yData)
- void [setData](#) (const QPolygonF &data)
- void [setData](#) (const [QwtData](#) &data)
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setPaintAttribute](#) ([PaintAttribute](#), bool on=true)
- void [setPen](#) (const QPen &)
- void [setRawData](#) (const double *x, const double *y, int size)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setStyle](#) ([CurveStyle](#) style)
- void [setSymbol](#) (const [QwtSymbol](#) &s)
- void [setTitle](#) (const QString &title)
- void [setTitle](#) (const [QwtText](#) &title)
- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setYAxis](#) (int axis)
- void [setZ](#) (double z)
- void [show](#) ()
- [CurveStyle](#) [style](#) () const
- const [QwtSymbol](#) & [symbol](#) () const
- bool [testCurveAttribute](#) ([CurveAttribute](#)) const
- bool [testItemAttribute](#) ([ItemAttribute](#)) const
- bool [testPaintAttribute](#) ([PaintAttribute](#)) const
- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- QRect [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const QwtDoubleRect &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- double [x](#) (int i) const
- int [xAxis](#) () const
- double [y](#) (int i) const
- int [yAxis](#) () const
- double [z](#) () const
- virtual ~[QwtPlotCurve](#) ()

Protected Member Functions

- void [closePolyline](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, [QwtPolygon](#) &) const
- virtual void [drawCurve](#) ([QPainter](#) *p, int style, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- void [drawDots](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- void [drawLines](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- void [drawSteps](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- void [drawSticks](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- virtual void [drawSymbols](#) ([QPainter](#) *p, const [QwtSymbol](#) &, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, int from, int to) const
- void [fillCurve](#) ([QPainter](#) *, const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, [QwtPolygon](#) &) const
- void [init](#) ()

12.51.1 Detailed Description

A plot item, that represents a series of points. A curve is the representation of a series of points in the x-y plane. It supports different display styles, interpolation (f.e. spline) and symbols.

Usage

- Assign curve properties** When a curve is created, it is configured to draw black solid lines with in Lines style and no symbols. You can change this by calling [setPen\(\)](#), [setStyle\(\)](#) and [setSymbol\(\)](#).
- Connect/Assign data.** [QwtPlotCurve](#) gets its points using a [QwtData](#) object offering a bridge to the real storage of the points (like [QAbstractItemModel](#)). There are several convenience classes derived from [QwtData](#), that also store the points inside (like [QStandardItemModel](#)). [QwtPlotCurve](#) also offers a couple of variations of [setData\(\)](#), that build [QwtData](#) objects from arrays internally.
- Attach the curve to a plot** See [QwtPlotItem::attach\(\)](#)

Example:

see examples/bode

See also

[QwtPlot](#), [QwtData](#), [QwtSymbol](#), [QwtScaleMap](#)

12.51.2 Member Enumeration Documentation

12.51.2.1 enum QwtPlotCurve::CurveAttribute

Attribute for drawing the curve

- Fitted (in combination with the Lines [QwtPlotCurve::CurveStyle](#) only)

A [QwtCurveFitter](#) tries to interpolate/smooth the curve, before it is painted. Note that curve fitting requires temporary memory for calculating coefficients and additional points. If painting in Fitted mode is slow it might be better to fit the points, before they are passed to [QwtPlotCurve](#).

- Inverted

For Steps only. Draws a step function from the right to the left.

See also

[setCurveAttribute\(\)](#), [testCurveAttribute\(\)](#), [curveFitter\(\)](#)

12.51.2.2 enum QwtPlotCurve::CurveStyle

Curve styles.

- 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\(\)](#).

- Sticks

Draw vertical(Yfx) or horizontal(Xfy) sticks from a baseline which is defined by [setBaseline\(\)](#).

- Steps

Connect the points with a step function. The step function is drawn from the left to the right or vice versa, depending on the 'Inverted' attribute.

- Dots

Draw dots at the locations of the data points. Note: This is different from a dotted line (see [setPen\(\)](#)), and faster as a curve in NoStyle style and a symbol painting a point.

- UserCurve

Styles \geq UserCurve are reserved for derived classes of [QwtPlotCurve](#) that overload [drawCurve\(\)](#) with additional application specific curve types.

See also

[setStyle\(\)](#), [style\(\)](#)

12.51.2.3 enum QwtPlotCurve::CurveType

Curve type.

- Yfx
Draws y as a function of x (the default). The baseline is interpreted as a horizontal line with y = [baseline\(\)](#).
- Xfy
Draws x as a function of y. The baseline is interpreted as a vertical line with x = [baseline\(\)](#).

The baseline is used for aligning the sticks, or filling the curve with a brush.

See also

[setCurveType\(\)](#), [curveType\(\)](#), [baseline\(\)](#) [brush\(\)](#)

12.51.2.4 enum QwtPlotItem::ItemAttribute [inherited]

Plot Item Attributes

- Legend
The item is represented on the legend.
- AutoScale
The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.51.2.5 enum QwtPlotCurve::PaintAttribute

Attributes to modify the drawing algorithm.

- PaintFiltered
Tries to reduce the data that has to be painted, by sorting out duplicates, or paintings outside the visible area. Might have a notable impact on curves with many close points. Only a couple of very basic filtering algos are implemented.
- ClipPolygons
Clip polygons before painting them. In situations, where points are far outside the visible area (f.e when zooming deep) this might be a substantial improvement for the painting performance (especially on Windows).

The default is, that no paint attributes are enabled.

See also

[setPaintAttribute\(\)](#), [testPaintAttribute\(\)](#)

12.51.2.6 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.51.2.7 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.51.3 Constructor & Destructor Documentation

12.51.3.1 QwtPlotCurve::QwtPlotCurve () [explicit]

Constructor.

12.51.3.2 QwtPlotCurve::QwtPlotCurve (const QwtText & *title*) [explicit]

Constructor

Parameters

<i>title</i>	Title of the curve
--------------	--------------------

12.51.3.3 QwtPlotCurve::QwtPlotCurve (const QString & *title*) [explicit]

Constructor

Parameters

<i>title</i>	Title of the curve
--------------	--------------------

12.51.3.4 QwtPlotCurve::~~QwtPlotCurve () [virtual]

Destructor.

12.51.4 Member Function Documentation**12.51.4.1 void QwtPlotItem::attach (QwtPlot * *plot*) [inherited]**

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.51.4.2 double QwtPlotCurve::baseline () const

Return the value of the baseline

See also

[setBaseline\(\)](#)

12.51.4.3 QwtDoubleRect QwtPlotCurve::boundingRect () const [virtual]

Returns the bounding rectangle of the curve data. If there is no bounding rect, like for empty data the rectangle is invalid.

See also

[QwtData::boundingRect\(\)](#), [QwtDoubleRect::isValid\(\)](#)

Reimplemented from [QwtPlotItem](#).

12.51.4.4 const QBrush & QwtPlotCurve::brush () const

Return the brush used to fill the area between lines and the baseline.

See also

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

12.51.4.5 `void QwtPlotCurve::closePolyline (const QwtScaleMap & xMap, const QwtScaleMap & yMap, QwtPolygon & pa) const`
[protected]

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

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>pa</i>	Polygon to be completed

12.51.4.6 `int QwtPlotCurve::closestPoint (const QPoint & pos, double * dist = NULL) const`

Find the closest curve point for a specific position

Parameters

<i>pos</i>	Position, where to look for the closest curve point
<i>dist</i>	If dist != NULL, closestPoint() returns the distance between the position and the closest curve point

Returns

Index of the closest curve point, or -1 if none can be found (f.e when the curve has no points)

Note

[closestPoint\(\)](#) implements a dumb algorithm, that iterates over all points

12.51.4.7 `QwtCurveFitter * QwtPlotCurve::curveFitter () const`

Get the curve fitter. If curve fitting is disabled NULL is returned.

Returns

Curve fitter

12.51.4.8 QwtPlotCurve::CurveType QwtPlotCurve::curveType () const

Return the curve type

See also

[CurveType](#), [setCurveType\(\)](#)

12.51.4.9 QwtData & QwtPlotCurve::data () [inline]

Returns

the the curve data

12.51.4.10 const QwtData & QwtPlotCurve::data () const [inline]

Returns

the the curve data

12.51.4.11 int QwtPlotCurve::dataSize () const

Return the size of the data arrays

See also

[setData\(\)](#)

12.51.4.12 void QwtPlotItem::detach () [inline, inherited]

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with.

[detach\(\)](#) is equivalent to calling [attach\(NULL \)](#)

See also

[attach\(QwtPlot* plot \)](#)

12.51.4.13 `void QwtPlotCurve::draw (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & canvasRect) const [virtual]`

Draw the complete curve.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	Maps x-values into pixel coordinates.
<i>yMap</i>	Maps y-values into pixel coordinates.

See also

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

Implements [QwtPlotItem](#).

12.51.4.14 `void QwtPlotCurve::draw (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const [virtual]`

Draw an interval of the curve.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	maps x-values into pixel coordinates.
<i>yMap</i>	maps y-values into pixel coordinates.
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted. If <i>to</i> < 0 the curve will be painted to its last point.

See also

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

12.51.4.15 `void QwtPlotCurve::draw (int from, int to) const`

Draw a set of points of a curve.

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

Setting `plot()->canvas()->setAttribute(Qt::WA_PaintOutsidePaintEvent, true);` will result in faster painting, if the paint engine of the canvas widget supports this feature.

Parameters

<i>from</i>	Index of the first point to be painted
<i>to</i>	Index of the last point to be painted. If <code>to < 0</code> the curve will be painted to its last point.

See also

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

12.51.4.16 `void QwtPlotCurve::drawCurve (QPainter * painter, int style, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const` `[protected, virtual]`

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

Parameters

<i>painter</i>	Painter
<i>style</i>	curve style, see QwtPlotCurve::CurveStyle
<i>xMap</i>	x map
<i>yMap</i>	y map
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted

See also

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

12.51.4.17 `void QwtPlotCurve::drawDots (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const` `[protected]`

Draw dots

Parameters

<i>painter</i>	Painter
<i>xMap</i>	x map
<i>yMap</i>	y map
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted

See also

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

12.51.4.18 `void QwtPlotCurve::drawLines (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const` **[protected]**

Draw lines.

If the CurveAttribute Fitted is enabled a [QwtCurveFitter](#) tries to interpolate/smooth the curve, before it is painted.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	x map
<i>yMap</i>	y map
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted

See also

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

12.51.4.19 `void QwtPlotCurve::drawSteps (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const` **[protected]**

Draw step function

The direction of the steps depends on Inverted attribute.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	x map
<i>yMap</i>	y map
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted

See also

[CurveAttribute](#), [setCurveAttribute\(\)](#), [draw\(\)](#), [drawCurve\(\)](#), [drawDots\(\)](#), [drawLines\(\)](#), [drawSticks\(\)](#)

12.51.4.20 `void QwtPlotCurve::drawSticks (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const` **[protected]**

Draw sticks

Parameters

<i>painter</i>	Painter
<i>xMap</i>	x map
<i>yMap</i>	y map
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted

See also

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

12.51.4.21 `void QwtPlotCurve::drawSymbols (QPainter * painter, const QwtSymbol & symbol, const QwtScaleMap & xMap, const QwtScaleMap & yMap, int from, int to) const` **[protected, virtual]**

Draw symbols.

Parameters

<i>painter</i>	Painter
<i>symbol</i>	Curve symbol
<i>xMap</i>	x map
<i>yMap</i>	y map
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted

See also

[setSymbol\(\)](#), [draw\(\)](#), [drawCurve\(\)](#)

12.51.4.22 `void QwtPlotCurve::fillCurve (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, QwtPolygon & pa) const` **[protected]**

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

Parameters

<i>painter</i>	Painter
<i>xMap</i>	x map

<i>yMap</i>	y map
<i>pa</i>	Polygon

See also

[setBrush\(\)](#), [setBaseline\(\)](#), [setCurveType\(\)](#)

12.51.4.23 void QwtPlotItem::hide () [inherited]

Hide the item.

12.51.4.24 void QwtPlotCurve::init () [protected]

Initialize data members.

12.51.4.25 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & rect) const [inherited]

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.51.4.26 bool QwtPlotItem::isVisible () const [inherited]**Returns**

true if visible

See also

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

12.51.4.27 void QwtPlotItem::itemChanged () [virtual, inherited]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.51.4.28 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.51.4.29 double QwtPlotCurve::maxXValue () const [inline]

[boundingRect\(\).right\(\)](#)

12.51.4.30 double QwtPlotCurve::maxYValue () const [inline]

[boundingRect\(\).bottom\(\)](#)

12.51.4.31 double QwtPlotCurve::minXValue () const [inline]

[boundingRect\(\).left\(\)](#)

12.51.4.32 `double QwtPlotCurve::minYValue () const [inline]`

[boundingRect\(\).top\(\)](#)

12.51.4.33 `QRect QwtPlotItem::paintRect (const QwtScaleMap & xMap,
const QwtScaleMap & yMap) const [inherited]`

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.51.4.34 `const QPen & QwtPlotCurve::pen () const`

Return the pen used to draw the lines.

See also

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

12.51.4.35 `QwtPlot * QwtPlotItem::plot () const [inherited]`

Return attached plot.

12.51.4.36 `int QwtPlotCurve::rtti () const [virtual]`**Returns**

QwtPlotItem::Rtti_PlotCurve

Reimplemented from [QwtPlotItem](#).

12.51.4.37 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const [inherited]

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.51.4.38 void QwtPlotItem::setAxis (int *xAxis*, int *yAxis*) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.51.4.39 void QwtPlotCurve::setBaseline (double *reference*)

Set the value of the baseline.

The baseline is needed for filling the curve with a brush or the Sticks drawing style. The default value is 0.0. The interpretation of the baseline depends on the CurveType. With QwtPlotCurve::Yfx, the baseline is interpreted as a horizontal line at `y = baseline\(\)`, with QwtPlotCurve::Yfy, it is interpreted as a vertical line at `x = baseline\(\)`.

Parameters

<i>reference</i>	baseline
------------------	----------

See also

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

12.51.4.40 void QwtPlotCurve::setBrush (const QBrush & *brush*)

Assign a brush.

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

In case `!brush.color().isValid()` the area will be filled by `pen.color()`. The fill algorithm simply connects the first and the last curve point to the baseline. So the curve data has to be sorted (ascending or descending).

Parameters

<i>brush</i>	New brush
--------------	-----------

See also

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

12.51.4.41 void QwtPlotCurve::setCurveAttribute (CurveAttribute *attribute*, bool *on* = *true*)

Specify an attribute for drawing the curve

Parameters

<i>attribute</i>	Curve attribute
<i>on</i>	On/Off

/sa CurveAttribute, [testCurveAttribute\(\)](#), [setCurveFitter\(\)](#)

12.51.4.42 void QwtPlotCurve::setCurveFitter (QwtCurveFitter * *curveFitter*)

Assign a curve fitter `setCurveFitter(NULL)` disables curve fitting.

Parameters

<i>curveFitter</i>	Curve fitter
--------------------	--------------

12.51.4.43 void QwtPlotCurve::setCurveType (CurveType *curveType*)

Assign the curve type

Parameters

<i>curveType</i>	Yfx or Xfy
------------------	------------

See also

[CurveType](#), [curveType\(\)](#)

12.51.4.44 void QwtPlotCurve::setData (const QPolygonF & *data*)

Initialize data with an array of points (explicitly shared).

Parameters

<i>data</i>	Data
-------------	------

Note

Internally the data is stored in a [QwtPolygonFData](#) object

12.51.4.45 void QwtPlotCurve::setData (const QwtArray< double > & *xData*, const QwtArray< double > & *yData*)

Initialize data with x- and y-arrays (explicitly shared) (Builds an [QwtArrayData](#) object internally)

Parameters

<i>xData</i>	x data
<i>yData</i>	y data

Note

Internally the data is stored in a [QwtArrayData](#) object

12.51.4.46 void QwtPlotCurve::setData (const QwtData & *data*)

Initialize data with a pointer to [QwtData](#).

Parameters

<i>data</i>	Data
-------------	------

See also

[QwtData::copy\(\)](#)

12.51.4.47 void QwtPlotCurve::setData (const double * *xData*, const double * *yData*, int *size*)

Set data by copying x- and y-values from specified memory blocks. Contrary to [setCurveRawData\(\)](#), this function makes a 'deep copy' of the data.

Parameters

<i>xData</i>	Pointer to x values
<i>yData</i>	Pointer to y values
<i>size</i>	Size of xData and yData

Note

Internally the data is stored in a [QwtArrayData](#) object

12.51.4.48 `void QwtPlotItem::setItemAttribute (ItemAttribute attribute, bool on = true)` [inherited]

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.51.4.49 `void QwtPlotCurve::setPaintAttribute (PaintAttribute attribute, bool on = true)`

Specify an attribute how to draw the curve

Parameters

<i>attribute</i>	Paint attribute
<i>on</i>	On/Off /sa PaintAttribute, testPaintAttribute()

12.51.4.50 `void QwtPlotCurve::setPen (const QPen & pen)`

Assign a pen

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

<i>pen</i>	New pen
------------	---------

See also

[pen\(\)](#), [brush\(\)](#), [QwtPainter::scaledPen\(\)](#)

12.51.4.51 void QwtPlotCurve::setRawData (const double * *xData*, const double * *yData*, int *size*)

Initialize the data by pointing to memory blocks which are not managed by [QwtPlotCurve](#).

setRawData is provided for efficiency. It is important to keep the pointers during the lifetime of the underlying [QwtCPointerData](#) class.

Parameters

<i>xData</i>	pointer to x data
<i>yData</i>	pointer to y data
<i>size</i>	size of x and y

Note

Internally the data is stored in a [QwtCPointerData](#) object

12.51.4.52 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on* = true) [inherited]

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.51.4.53 void QwtPlotCurve::setStyle (CurveStyle *style*)

Set the curve's drawing style

Parameters

<i>style</i>	Curve style
--------------	-------------

See also

[CurveStyle](#), [style\(\)](#)

12.51.4.54 void QwtPlotCurve::setSymbol (const QwtSymbol & *symbol*)

Assign a symbol.

Parameters

<i>symbol</i>	Symbol
---------------	--------

See also

[symbol\(\)](#)

**12.51.4.55 void QwtPlotItem::setTitle (const QwtText & *title*)
[inherited]**

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

**12.51.4.56 void QwtPlotItem::setTitle (const QString & *title*)
[inherited]**

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

**12.51.4.57 void QwtPlotItem::setVisible (bool *on*) [virtual,
inherited]**

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also

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

12.51.4.58 void QwtPlotItem::setXAxis (int *axis*) [inherited]

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also

[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)

12.51.4.59 void QwtPlotItem::setYAxis (int *axis*) [inherited]

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also

[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)

12.51.4.60 void QwtPlotItem::setZ (double *z*) [inherited]

Set the *z* value.

Plot items are painted in increasing *z*-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also

[z\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.51.4.61 void QwtPlotItem::show () [inherited]

Show the item.

12.51.4.62 QwtPlotCurve::CurveStyle QwtPlotCurve::style () const

Return the current style

See also

[CurveStyle](#), [setStyle\(\)](#)

12.51.4.63 const QwtSymbol & QwtPlotCurve::symbol () const

Return the current symbol.

See also

[setSymbol\(\)](#)

12.51.4.64 bool QwtPlotCurve::testCurveAttribute (CurveAttribute *attribute*) const

Returns

true, if attribute is enabled

See also

[CurveAttribute](#), [setCurveAttribute\(\)](#)

12.51.4.65 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*) const [inherited]

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also

[setItemAttribute\(\)](#), [ItemAttribute](#)

12.51.4.66 `bool QwtPlotCurve::testPaintAttribute (PaintAttribute attribute) const`

Return the current paint attributes.

See also

[PaintAttribute](#), [setPaintAttribute\(\)](#)

12.51.4.67 `bool QwtPlotItem::testRenderHint (RenderHint hint) const`
[[inherited](#)]

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.51.4.68 `const QwtText & QwtPlotItem::title () const` [[inherited](#)]

Returns

Title of the item

See also

[setTitle\(\)](#)

12.51.4.69 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap,
const QwtScaleMap & yMap, const QwtDoubleRect & rect) const`
[[inherited](#)]

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.51.4.70 `void QwtPlotCurve::updateLegend (QwtLegend * legend) const`
[virtual]

Update the widget that represents the curve on the legend.

Reimplemented from [QwtPlotItem](#).

12.51.4.71 `void QwtPlotItem::updateScaleDiv (const QwtScaleDiv &, const`
`QwtScaleDiv &) [virtual, inherited]`

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 [QwtPlotGrid\(\)](#)) have to reimplement [updateScaleDiv\(\)](#)

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented in [QwtPlotGrid](#), and [QwtPlotScaleItem](#).

12.51.4.72 `double QwtPlotCurve::x (int i) const` **[inline]**

Parameters

<i>i</i>	index
----------	-------

Returns

x-value at position *i*

12.51.4.73 `int QwtPlotItem::xAxis () const` `[inherited]`

Return xAxis.

12.51.4.74 `double QwtPlotCurve::y (int i) const` `[inline]`

Parameters

<i>i</i> index

Returns

y-value at position *i*

12.51.4.75 `int QwtPlotItem::yAxis () const` `[inherited]`

Return yAxis.

12.51.4.76 `double QwtPlotItem::z () const` `[inherited]`

Plot items are painted in increasing z-order.

Returns

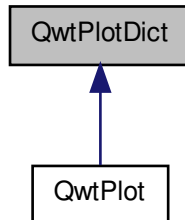
[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.52 QwtPlotDict Class Reference

A dictionary for plot items.

```
#include <qwt_plot_dict.h>
```

Inheritance diagram for QwtPlotDict:



Public Member Functions

- bool [autoDelete](#) () const
- void [detachItems](#) (int rtti=QwtPlotItem::Rtti_PlotItem, bool autoDelete=true)
- const QwtPlotItemList & [itemList](#) () const
- [QwtPlotDict](#) ()
- void [setAutoDelete](#) (bool)
- [~QwtPlotDict](#) ()

Friends

- class [QwtPlotItem](#)

12.52.1 Detailed Description

A dictionary for plot items. [QwtPlotDict](#) organizes plot items in increasing z-order. If [autoDelete\(\)](#) is enabled, all attached items will be deleted in the destructor of the dictionary.

See also

[QwtPlotItem::attach\(\)](#), [QwtPlotItem::detach\(\)](#), [QwtPlotItem::z\(\)](#)

12.52.2 Constructor & Destructor Documentation

12.52.2.1 QwtPlotDict::QwtPlotDict () [**explicit**]

Constructor

Auto deletion is enabled.

See also

[setAutoDelete\(\)](#), [attachItem\(\)](#)

12.52.2.2 QwtPlotDict::~~QwtPlotDict ()

Destructor

If autoDelete is on, all attached items will be deleted

See also

[setAutoDelete\(\)](#), [autoDelete\(\)](#), [attachItem\(\)](#)

12.52.3 Member Function Documentation**12.52.3.1 bool QwtPlotDict::autoDelete () const****Returns**

true if auto deletion is enabled

See also

[setAutoDelete\(\)](#), [attachItem\(\)](#)

12.52.3.2 void QwtPlotDict::detachItems (int rtti = *QwtPlotItem::Rtti_PlotItem*, bool autoDelete = true)

Detach items from the dictionary

Parameters

<i>rtti</i>	In case of <code>QwtPlotItem::Rtti_PlotItem</code> detach all items otherwise only those items of the type <code>rtti</code> .
<i>autoDelete</i>	If true, delete all detached items

12.52.3.3 const QwtPlotItemList & QwtPlotDict::itemList () const

A `QwtPlotItemList` of all attached plot items.

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

Returns

List of all attached plot items.

12.52.3.4 void QwtPlotDict::setAutoDelete (bool *autoDelete*)

En/Disable Auto deletion

If Auto deletion is on all attached plot items will be deleted in the destructor of [QwtPlotDict](#). The default value is on.

See also

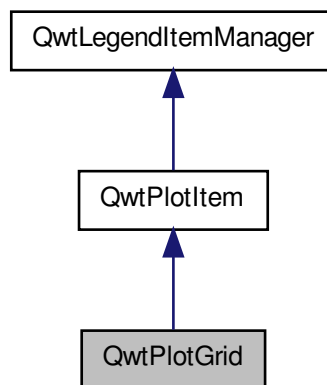
[autoDelete\(\)](#), [attachItem\(\)](#)

12.53 QwtPlotGrid Class Reference

A class which draws a coordinate grid.

```
#include <qwt_plot_grid.h>
```

Inheritance diagram for QwtPlotGrid:

**Public Types**

- enum [ItemAttribute](#) {
 Legend = 1,
 AutoScale = 2 }

- enum [RenderHint](#) { **RenderAntialiased** = 1 }
- enum [RttiValues](#) {
Rtti_PlotItem = 0,
Rtti_PlotGrid,
Rtti_PlotScale,
Rtti_PlotMarker,
Rtti_PlotCurve,
Rtti_PlotHistogram,
Rtti_PlotSpectrogram,
Rtti_PlotSVG,
Rtti_PlotUserItem = 1000 }

Public Member Functions

- void [attach](#) ([QwtPlot](#) *plot)
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &rect) const
- void [enableX](#) (bool tf)
- void [enableXMin](#) (bool tf)
- void [enableY](#) (bool tf)
- void [enableYMin](#) (bool tf)
- void [hide](#) ()
- [QwtDoubleRect](#) [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QRect](#) &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual [QWidget](#) * [legendItem](#) () const
- const [QPen](#) & [majPen](#) () const
- const [QPen](#) & [minPen](#) () const
- [QRect](#) [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- [QwtPlot](#) * [plot](#) () const
- [QwtPlotGrid](#) ()
- virtual int [rtti](#) () const
- [QwtDoubleRect](#) [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAxis](#) (int xAxis, int yAxis)
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setMajPen](#) (const [QPen](#) &p)
- void [setMinPen](#) (const [QPen](#) &p)
- void [setPen](#) (const [QPen](#) &p)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setTitle](#) (const [QString](#) &title)
- void [setTitle](#) (const [QwtText](#) &title)

- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setXDiv](#) (const [QwtScaleDiv](#) &sx)
- void [setYAxis](#) (int axis)
- void [setYDiv](#) (const [QwtScaleDiv](#) &sy)
- void [setZ](#) (double z)
- void [show](#) ()
- bool [testItemAttribute](#) ([ItemAttribute](#)) const
- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- QRect [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const Qwt-DoubleRect &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &xMap, const [QwtScaleDiv](#) &yMap)
- int [xAxis](#) () const
- bool [xEnabled](#) () const
- bool [xMinEnabled](#) () const
- const [QwtScaleDiv](#) & [xScaleDiv](#) () const
- int [yAxis](#) () const
- bool [yEnabled](#) () const
- bool [yMinEnabled](#) () const
- const [QwtScaleDiv](#) & [yScaleDiv](#) () const
- double [z](#) () const
- virtual [~QwtPlotGrid](#) ()

12.53.1 Detailed Description

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

12.53.2 Member Enumeration Documentation

12.53.2.1 enum [QwtPlotItem::ItemAttribute](#) [\[inherited\]](#)

Plot Item Attributes

- Legend
The item is represented on the legend.
- AutoScale
The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.53.2.2 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.53.2.3 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.53.3 Constructor & Destructor Documentation

12.53.3.1 QwtPlotGrid::QwtPlotGrid () [explicit]

Enables major grid, disables minor grid.

12.53.3.2 QwtPlotGrid::~QwtPlotGrid () [virtual]

Destructor.

12.53.4 Member Function Documentation

12.53.4.1 void QwtPlotItem::attach (QwtPlot * *plot*) [inherited]

Attach the item to a plot.

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

Parameters

<i>plot</i> Plot widget

See also

[QwtPlotItem::detach\(\)](#)

12.53.4.2 `QwtDoubleRect QwtPlotItem::boundingRect () const`
[virtual, inherited]

Returns

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

Reimplemented in [QwtPlotCurve](#), [QwtPlotMarker](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.53.4.3 `void QwtPlotItem::detach () [inline, inherited]`

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with. [detach\(\)](#) is equivalent to calling `attach(NULL)`

See also

[attach\(QwtPlot* plot \)](#)

12.53.4.4 `void QwtPlotGrid::draw (QPainter *painter, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &canvasRect)`
const [virtual]

Draw the grid.

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

Parameters

<i>painter</i>	Painter
<i>xMap</i>	X axis map
<i>yMap</i>	Y axis
<i>canvasRect</i>	Contents rect of the plot canvas

Implements [QwtPlotItem](#).

12.53.4.5 void QwtPlotGrid::enableX (bool *tf*)

Enable or disable vertical gridlines.

Parameters

<i>tf</i> Enable (true) or disable

See also

Minor gridlines can be enabled or disabled with [enableXMin\(\)](#)

12.53.4.6 void QwtPlotGrid::enableXMin (bool *tf*)

Enable or disable minor vertical gridlines.

Parameters

<i>tf</i> Enable (true) or disable

See also

[enableX\(\)](#)

12.53.4.7 void QwtPlotGrid::enableY (bool *tf*)

Enable or disable horizontal gridlines.

Parameters

<i>tf</i> Enable (true) or disable

See also

Minor gridlines can be enabled or disabled with [enableYMin\(\)](#)

12.53.4.8 void QwtPlotGrid::enableYMin (bool *tf*)

Enable or disable minor horizontal gridlines.

Parameters

<i>tf</i> Enable (true) or disable

See also

[enableY\(\)](#)

12.53.4.9 void QwtPlotItem::hide () `[inherited]`

Hide the item.

12.53.4.10 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *rect*) const `[inherited]`

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.53.4.11 bool QwtPlotItem::isVisible () const `[inherited]`

Returns

true if visible

See also

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

12.53.4.12 void QwtPlotItem::itemChanged () `[virtual, inherited]`

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.53.4.13 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.53.4.14 const QPen & QwtPlotGrid::majPen () const**Returns**

the pen for the major gridlines

See also

[setMajPen\(\)](#), [setMinPen\(\)](#), [setPen\(\)](#)

12.53.4.15 const QPen & QwtPlotGrid::minPen () const**Returns**

the pen for the minor gridlines

See also

[setMinPen\(\)](#), [setMajPen\(\)](#), [setPen\(\)](#)

12.53.4.16 QRect QwtPlotItem::paintRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const [inherited]

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.53.4.17 QwtPlot * QwtPlotItem::plot () const [inherited]

Return attached plot.

12.53.4.18 int QwtPlotGrid::rtti () const [virtual]**Returns**

QwtPlotItem::Rtti_PlotGrid

Reimplemented from [QwtPlotItem](#).

12.53.4.19 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.53.4.20 void QwtPlotItem::setAxis (int xAxis, int yAxis) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.53.4.21 `void QwtPlotItem::setItemAttribute (ItemAttribute attribute, bool on = true) [inherited]`

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.53.4.22 `void QwtPlotGrid::setMajPen (const QPen & pen)`

Assign a pen for the major gridlines

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

<i>pen</i>	Pen
------------	-----

See also

[majPen\(\)](#), [setMinPen\(\)](#), [setPen\(\)](#), [QwtPainter::scaledPen\(\)](#)

12.53.4.23 `void QwtPlotGrid::setMinPen (const QPen & pen)`

Assign a pen for the minor gridlines

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

<i>pen</i>	Pen
------------	-----

See also

[minPen\(\)](#), [setMajPen\(\)](#), [setPen\(\)](#), [QwtPainter::scaledPen\(\)](#)

12.53.4.24 `void QwtPlotGrid::setPen (const QPen & pen)`

Assign a pen for both major and minor gridlines

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

<i>pen</i>	Pen
------------	-----

See also

[setMajPen\(\)](#), [setMinPen\(\)](#), [QwtPainter::scaledPen\(\)](#)

12.53.4.25 `void QwtPlotItem::setRenderHint (RenderHint hint, bool on = true)` `[inherited]`

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.53.4.26 `void QwtPlotItem::setTitle (const QwtText & title)` `[inherited]`

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.53.4.27 `void QwtPlotItem::setTitle (const QString & title)` `[inherited]`

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.53.4.28 void QwtPlotItem::setVisible (bool *on*) [virtual, inherited]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also[isVisible\(\)](#), [show\(\)](#), [hide\(\)](#)**12.53.4.29 void QwtPlotItem::setXAxis (int *axis*) [inherited]**

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)**12.53.4.30 void QwtPlotGrid::setXDiv (const QwtScaleDiv & *scaleDiv*)**

Assign an x axis scale division

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

12.53.4.31 void QwtPlotItem::setYAxis (int *axis*) [inherited]

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)

12.53.4.32 void QwtPlotGrid::setYDiv (const QwtScaleDiv & *scaleDiv*)

Assign a y axis division

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

12.53.4.33 void QwtPlotItem::setZ (double *z*) [inherited]

Set the z value.

Plot items are painted in increasing z-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also[z\(\)](#), [QwtPlotDict::itemList\(\)](#)**12.53.4.34 void QwtPlotItem::show () [inherited]**

Show the item.

**12.53.4.35 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*)
const [inherited]**

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also[setItemAttribute\(\)](#), [ItemAttribute](#)

12.53.4.36 `bool QwtPlotItem::testRenderHint (RenderHint hint) const`
[[inherited](#)]

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.53.4.37 `const QwtText & QwtPlotItem::title () const` [[inherited](#)]

Returns

Title of the item

See also

[setTitle\(\)](#)

12.53.4.38 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap,
const QwtScaleMap & yMap, const QwtDoubleRect & rect) const`
[[inherited](#)]

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.53.4.39 void QwtPlotItem::updateLegend (QwtLegend * *legend*) const [virtual, inherited]

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also

[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

12.53.4.40 void QwtPlotGrid::updateScaleDiv (const QwtScaleDiv & *xScaleDiv*, const QwtScaleDiv & *yScaleDiv*) [virtual]

Update the grid to changes of the axes scale division

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented from [QwtPlotItem](#).

12.53.4.41 int QwtPlotItem::xAxis () const [inherited]

Return xAxis.

12.53.4.42 bool QwtPlotGrid::xEnabled () const

Returns

true if vertical gridlines are enabled

See also

[enableX\(\)](#)

12.53.4.43 bool QwtPlotGrid::xMinEnabled () const**Returns**

true if minor vertical gridlines are enabled

See also

[enableXMin\(\)](#)

12.53.4.44 const QwtScaleDiv & QwtPlotGrid::xScaleDiv () const**Returns**

the scale division of the x axis

12.53.4.45 int QwtPlotItem::yAxis () const [inherited]

Return yAxis.

12.53.4.46 bool QwtPlotGrid::yEnabled () const**Returns**

true if horizontal gridlines are enabled

See also

[enableY\(\)](#)

12.53.4.47 bool QwtPlotGrid::yMinEnabled () const**Returns**

true if minor horizontal gridlines are enabled

See also

[enableYMin\(\)](#)

12.53.4.48 const QwtScaleDiv & QwtPlotGrid::yScaleDiv () const**Returns**

the scale division of the y axis

12.53.4.49 double QwtPlotItem::z () const [inherited]

Plot items are painted in increasing z-order.

Returns

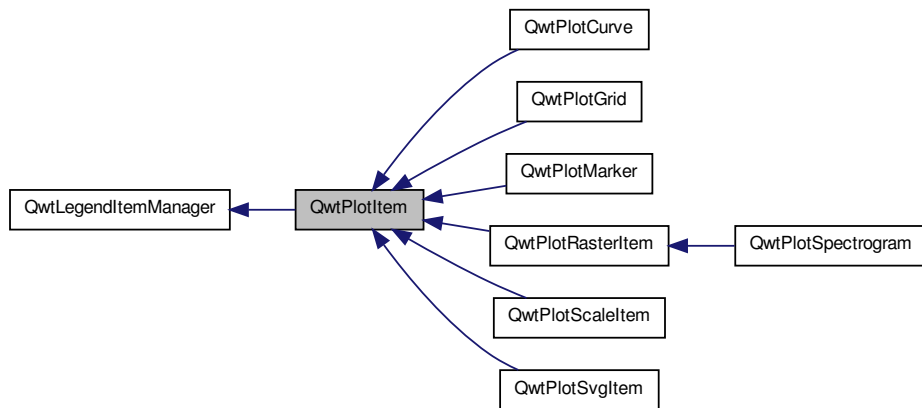
[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.54 QwtPlotItem Class Reference

Base class for items on the plot canvas.

```
#include <qwt_plot_item.h>
```

Inheritance diagram for QwtPlotItem:



Public Types

- enum [ItemAttribute](#) {
Legend = 1,
AutoScale = 2 }
- enum [RenderHint](#) { **RenderAntialiased** = 1 }
- enum [RttiValues](#) {
Rtti_PlotItem = 0,
Rtti_PlotGrid,
Rtti_PlotScale,
Rtti_PlotMarker,
Rtti_PlotCurve,
Rtti_PlotHistogram,
Rtti_PlotSpectrogram,
Rtti_PlotSVG,
Rtti_PlotUserItem = 1000 }

Public Member Functions

- void [attach](#) ([QwtPlot](#) *plot)
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *painter, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &canvasRect) const =0

- void [hide](#) ()
- QwtDoubleRect [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const QRect &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual QWidget * [legendItem](#) () const
- QRect [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- [QwtPlot](#) * [plot](#) () const
- [QwtPlotItem](#) (const [QwtText](#) &title=[QwtText](#)())
- virtual int [rtti](#) () const
- QwtDoubleRect [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAxis](#) (int xAxis, int yAxis)
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setTitle](#) (const [QwtText](#) &title)
- void [setTitle](#) (const QString &title)
- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setYAxis](#) (int axis)
- void [setZ](#) (double z)
- void [show](#) ()
- bool [testItemAttribute](#) ([ItemAttribute](#)) const
- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- QRect [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const QwtDoubleRect &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- int [xAxis](#) () const
- int [yAxis](#) () const
- double [z](#) () const
- virtual ~[QwtPlotItem](#) ()

12.54.1 Detailed Description

Base class for items on the plot canvas. A plot item is "something", that can be painted on the plot canvas, or only affects the scales of the plot widget. They can be categorized as:

- Representator

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

 - [QwtPlotMarker](#) Represents a point or a horizontal/vertical coordinate
 - [QwtPlotCurve](#) Represents a series of points

- [QwtPlotSpectrogram](#) ([QwtPlotRasterItem](#)) Represents raster data
- ...

- Decorators

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

- [QwtPlotGrid](#)
- [QwtPlotScaleItem](#)
- [QwtPlotSvgItem](#)
- ...

Depending on the [QwtPlotItem::ItemAttribute](#) flags, an item is included into autoscaling or has an entry on the legend.

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

See also

The `cpuplot` example shows the implementation of additional [plot](#) items.

12.54.2 Member Enumeration Documentation

12.54.2.1 enum QwtPlotItem::ItemAttribute

Plot Item Attributes

- Legend

The item is represented on the legend.

- AutoScale

The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.54.2.2 enum QwtPlotItem::RenderHint

Render hints.

12.54.2.3 enum QwtPlotItem::RttiValues

Runtime type information.

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

12.54.3 Constructor & Destructor Documentation

12.54.3.1 QwtPlotItem::QwtPlotItem (const QwtText & *title* = QwtText ()) [explicit]

Constructor

Parameters

<i>title</i>	Title of the item
--------------	-------------------

12.54.3.2 QwtPlotItem::~QwtPlotItem () [virtual]

Destroy the [QwtPlotItem](#).

12.54.4 Member Function Documentation

12.54.4.1 void QwtPlotItem::attach (QwtPlot * *plot*)

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.54.4.2 QwtDoubleRect QwtPlotItem::boundingRect () const [virtual]

Returns

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

Reimplemented in [QwtPlotCurve](#), [QwtPlotMarker](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.54.4.3 void QwtPlotItem::detach () [inline]

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with. [detach\(\)](#) is equivalent to calling [attach\(NULL \)](#)

See also

[attach\(QwtPlot* plot \)](#)

12.54.4.4 virtual void QwtPlotItem::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [pure virtual]

Draw the item.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	Maps x-values into pixel coordinates.
<i>yMap</i>	Maps y-values into pixel coordinates.
<i>canvasRect</i>	Contents rect of the canvas in painter coordinates

Implemented in [QwtPlotCurve](#), [QwtPlotGrid](#), [QwtPlotMarker](#), [QwtPlotRasterItem](#), [QwtPlotScaleItem](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.54.4.5 void QwtPlotItem::hide ()

Hide the item.

12.54.4.6 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *rect*) const

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.54.4.7 bool QwtPlotItem::isVisible () const**Returns**

true if visible

See also

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

12.54.4.8 void QwtPlotItem::itemChanged () [virtual]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.54.4.9 QWidget * QwtPlotItem::legendItem () const [virtual]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.54.4.10 QRect QwtPlotItem::paintRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.54.4.11 QwtPlot * QwtPlotItem::plot () const

Return attached plot.

12.54.4.12 int QwtPlotItem::rtti () const [virtual]

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

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 [QwtPlotCurve](#), [QwtPlotGrid](#), [QwtPlotMarker](#), [QwtPlotScaleItem](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.54.4.13 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.54.4.14 void QwtPlotItem::setAxis (int *xAxis*, int *yAxis*)

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.54.4.15 void QwtPlotItem::setItemAttribute (ItemAttribute *attribute*, bool *on = true*)

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.54.4.16 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on = true*)

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.54.4.17 void QwtPlotItem::setTitle (const QString & *title*)

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.54.4.18 void QwtPlotItem::setTitle (const QwtText & *title*)

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.54.4.19 void QwtPlotItem::setVisible (bool *on*) [virtual]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also

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

12.54.4.20 void QwtPlotItem::setXAxis (int *axis*)

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also

[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)

12.54.4.21 void QwtPlotItem::setYAxis (int *axis*)

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also

[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)

12.54.4.22 void QwtPlotItem::setZ (double *z*)

Set the *z* value.

Plot items are painted in increasing *z*-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also

[z\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.54.4.23 void QwtPlotItem::show ()

Show the item.

12.54.4.24 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*) const

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also

[setItemAttribute\(\)](#), [ItemAttribute](#)

12.54.4.25 bool QwtPlotItem::testRenderHint (RenderHint *hint*) const

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.54.4.26 const QwtText & QwtPlotItem::title () const**Returns**

Title of the item

See also

[setTitle\(\)](#)

**12.54.4.27 QRect QwtPlotItem::transform (const QwtScaleMap & *xMap*,
const QwtScaleMap & *yMap*, const QwtDoubleRect & *rect*) const**

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also[invTransform\(\)](#)**12.54.4.28 void QwtPlotItem::updateLegend (QwtLegend * *legend*) const
[virtual]**

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

**12.54.4.29 void QwtPlotItem::updateScaleDiv (const QwtScaleDiv & , const
QwtScaleDiv &) [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 [QwtPlotGrid\(\)](#)) have to reimplement [updateScaleDiv\(\)](#)

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also[QwtPlot::updateAxes\(\)](#)

Reimplemented in [QwtPlotGrid](#), and [QwtPlotScaleItem](#).

12.54.4.30 int QwtPlotItem::xAxis () const

Return xAxis.

12.54.4.31 int QwtPlotItem::yAxis () const

Return yAxis.

12.54.4.32 double QwtPlotItem::z () const

Plot items are painted in increasing z-order.

Returns

[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.55 QwtPlotLayout Class Reference

Layout engine for [QwtPlot](#).

```
#include <qwt_plot_layout.h>
```

Public Types

- enum [Options](#) {
 AlignScales = 1,
 IgnoreScrollbars = 2,
 IgnoreFrames = 4,
 IgnoreMargin = 8,
 IgnoreLegend = 16 }

Public Member Functions

- virtual void [activate](#) (const [QwtPlot](#) *, const QRect &rect, int options=0)
- bool [alignCanvasToScales](#) () const
- int [canvasMargin](#) (int axis) const
- const QRect & [canvasRect](#) () const
- virtual void [invalidate](#) ()
- [QwtPlot::LegendPosition](#) [legendPosition](#) () const
- double [legendRatio](#) () const
- const QRect & [legendRect](#) () const
- int [margin](#) () const

- virtual QSize [minimumSizeHint](#) (const [QwtPlot](#) *) const
- [QwtPlotLayout](#) ()
- const QRect & [scaleRect](#) (int axis) const
- void [setAlignCanvasToScales](#) (bool)
- void [setCanvasMargin](#) (int margin, int axis=-1)
- void [setLegendPosition](#) ([QwtPlot::LegendPosition](#) pos, double ratio)
- void [setLegendPosition](#) ([QwtPlot::LegendPosition](#) pos)
- void [setLegendRatio](#) (double ratio)
- void [setMargin](#) (int)
- void [setSpacing](#) (int)
- int [spacing](#) () const
- const QRect & [titleRect](#) () const
- virtual ~[QwtPlotLayout](#) ()

Protected Member Functions

- QRect [alignLegend](#) (const QRect &canvasRect, const QRect &legendRect) const
- void [alignScales](#) (int options, QRect &canvasRect, QRect scaleRect[[QwtPlot::axisCnt](#)]) const
- void [expandLineBreaks](#) (int options, const QRect &rect, int &dimTitle, int dimAxes[[QwtPlot::axisCnt](#)]) const
- QRect [layoutLegend](#) (int options, const QRect &) const

12.55.1 Detailed Description

Layout engine for [QwtPlot](#). It is used by the [QwtPlot](#) widget to organize its internal widgets or by [QwtPlot::print\(\)](#) to render its content to a QPaintDevice like a QPrinter, QPixmap/QImage or QSvgRenderer.

12.55.2 Member Enumeration Documentation

12.55.2.1 enum [QwtPlotLayout::Options](#)

Options to configure the plot layout engine

- AlignScales
Unused
- IgnoreScrollbars
Ignore the dimension of the scrollbars. There are no scrollbars, when the plot is rendered to a paint device ([QwtPlot::print\(\)](#)).
- IgnoreFrames
Ignore all frames. [QwtPlot::print\(\)](#) doesn't paint them.
- IgnoreMargin
Ignore the [margin\(\)](#).

- IgnoreLegend
Ignore the legend.

See also

[activate\(\)](#)

12.55.3 Constructor & Destructor Documentation**12.55.3.1 QwtPlotLayout::QwtPlotLayout () [explicit]**

Constructor.

12.55.3.2 QwtPlotLayout::~~QwtPlotLayout () [virtual]

Destructor.

12.55.4 Member Function Documentation**12.55.4.1 void QwtPlotLayout::activate (const QwtPlot * *plot*, const QRect & *plotRect*, int *options* = 0) [virtual]**

Recalculate the geometry of all components.

Parameters

<i>plot</i>	Plot to be layout
<i>plotRect</i>	Rect where to place the components
<i>options</i>	Options

See also

[invalidate\(\)](#), [Options](#), [titleRect\(\)](#), [legendRect\(\)](#), [scaleRect\(\)](#), [canvasRect\(\)](#)

12.55.4.2 bool QwtPlotLayout::alignCanvasToScales () const

Return the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size
- align with the axis scale ends to control its size.

Returns

align-canvas-to-axis-scales setting

See also

[setAlignCanvasToScales](#), [setCanvasMargin\(\)](#)

Note

In this context the term 'scale' means the backbone of a scale.

12.55.4.3 `QRect QwtPlotLayout::alignLegend (const QRect & canvasRect,
const QRect & legendRect) const` **[protected]**

Align the legend to the canvas

Parameters

<i>canvasRect</i>	Geometry of the canvas
<i>legendRect</i>	Maximum geometry for the legend

Returns

Geometry for the aligned legend

12.55.4.4 `void QwtPlotLayout::alignScales (int options, QRect & canvasRect,
QRect scaleRect[QwtPlot::axisCnt]) const` **[protected]**

Align the ticks of the axis to the canvas borders using the empty corners.

See also

[Options](#)

12.55.4.5 `int QwtPlotLayout::canvasMargin (int axis) const`

Returns

Margin around the scale tick borders

See also

[setCanvasMargin\(\)](#)

12.55.4.6 `const QRect & QwtPlotLayout::canvasRect () const`**Returns**

Geometry for the canvas

See also

[activate\(\)](#), [invalidate\(\)](#)

12.55.4.7 `void QwtPlotLayout::expandLineBreaks (int options, const QRect & rect, int & dimTitle, int dimAxis[QwtPlot::axisCnt]) const`
[protected]

Expand all line breaks in text labels, and calculate the height of their widgets in orientation of the text.

Parameters

<i>options</i>	Options how to layout the legend
<i>rect</i>	Bounding rect for title, axes and canvas.
<i>dimTitle</i>	Expanded height of the title widget
<i>dimAxis</i>	Expanded heights of the axis in axis orientation.

See also

[Options](#)

12.55.4.8 `void QwtPlotLayout::invalidate ()` [virtual]

Invalidate the geometry of all components.

See also

[activate\(\)](#)

12.55.4.9 `QRect QwtPlotLayout::layoutLegend (int options, const QRect & rect) const` [protected]

Find the geometry for the legend

Parameters

<i>options</i>	Options how to layout the legend
<i>rect</i>	Rectangle where to place the legend

Returns

Geometry for the legend

See also

[Options](#)

12.55.4.10 QwtPlot::LegendPosition QwtPlotLayout::legendPosition () const**Returns**

Position of the legend

See also

[setLegendPosition\(\)](#), [QwtPlot::setLegendPosition\(\)](#), [QwtPlot::legendPosition\(\)](#)

12.55.4.11 double QwtPlotLayout::legendRatio () const**Returns**

The relative size of the legend in the plot.

See also

[setLegendPosition\(\)](#)

12.55.4.12 const QRect & QwtPlotLayout::legendRect () const**Returns**

Geometry for the legend

See also

[activate\(\)](#), [invalidate\(\)](#)

12.55.4.13 int QwtPlotLayout::margin () const

Returns

margin

See also[setMargin\(\)](#), [spacing\(\)](#), [QwtPlot::margin\(\)](#)**12.55.4.14 QSize QwtPlotLayout::minimumSizeHint (const QwtPlot * *plot*) const [virtual]**

Return a minimum size hint.

See also[QwtPlot::minimumSizeHint\(\)](#)**12.55.4.15 const QRect & QwtPlotLayout::scaleRect (int *axis*) const****Parameters**

<i>axis</i>	Axis index
-------------	------------

Returns

Geometry for the scale

See also[activate\(\)](#), [invalidate\(\)](#)**12.55.4.16 void QwtPlotLayout::setAlignCanvasToScales (bool *alignCanvasToScales*)**

Change the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size,
- align with the axis scale ends to control its size.

Parameters

<i>alignCanvasToScales</i>	New align-canvas-to-axis-scales setting
----------------------------	---

See also[setCanvasMargin\(\)](#)**Note**

In this context the term 'scale' means the backbone of a scale.

Warning

In case of `alignCanvasToScales == true` `canvasMargin` will have no effect

12.55.4.17 void QwtPlotLayout::setCanvasMargin (int *margin*, int *axis* = -1)

Change a margin of the canvas. The margin is the space above/below the scale ticks. A negative margin will be set to -1, excluding the borders of the scales.

Parameters

<i>margin</i>	New margin
<i>axis</i>	One of QwtPlot::Axis . Specifies where the position of the margin. -1 means margin at all borders.

See also[canvasMargin\(\)](#)**Warning**

The margin will have no effect when `alignCanvasToScales` is true

12.55.4.18 void QwtPlotLayout::setLegendPosition (QwtPlot::LegendPosition *pos*, double *ratio*)

Specify the position of the legend.

Parameters

<i>pos</i>	The legend's position.
<i>ratio</i>	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[QwtPlot::setLegendPosition\(\)](#)

12.55.4.19 void QwtPlotLayout::setLegendPosition (QwtPlot::LegendPosition *pos*)

Specify the position of the legend.

Parameters

<i>pos</i>	The legend's position. Valid values are QwtPlot::LeftLegend, QwtPlot::RightLegend, QwtPlot::TopLegend, QwtPlot::BottomLegend.
------------	---

See also

QwtPlot::setLegendPosition()

12.55.4.20 void QwtPlotLayout::setLegendRatio (double *ratio*)

Specify the relative size of the legend in the plot

Parameters

<i>ratio</i>	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of ≤ 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.
--------------	---

12.55.4.21 void QwtPlotLayout::setMargin (int *margin*)

Change the margin of the plot. The margin is the space around all components.

Parameters

<i>margin</i>	new margin
---------------	------------

See also

[margin\(\)](#), [setSpacing\(\)](#), [QwtPlot::setMargin\(\)](#)

12.55.4.22 void QwtPlotLayout::setSpacing (int *spacing*)

Change the spacing of the plot. The spacing is the distance between the plot components.

Parameters

<i>spacing</i>	new spacing
----------------	-------------

See also

[setMargin\(\)](#), [spacing\(\)](#)

12.55.4.23 int QwtPlotLayout::spacing () const

Returns

spacing

See also

[margin\(\)](#), [setSpacing\(\)](#)

12.55.4.24 const QRect & QwtPlotLayout::titleRect () const

Returns

Geometry for the title

See also

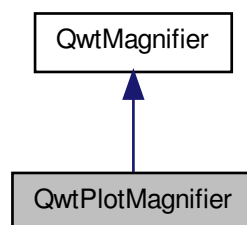
[activate\(\)](#), [invalidate\(\)](#)

12.56 QwtPlotMagnifier Class Reference

[QwtPlotMagnifier](#) provides zooming, by magnifying in steps.

```
#include <qwt_plot_magnifier.h>
```

Inheritance diagram for QwtPlotMagnifier:



Public Member Functions

- [QwtPlotCanvas * canvas](#) ()
- const [QwtPlotCanvas * canvas](#) () const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [getMouseButton](#) (int &button, int &buttonState) const
- void [getZoomInKey](#) (int &key, int &modifiers) const
- void [getZoomOutKey](#) (int &key, int &modifiers) const
- bool [isAxisEnabled](#) (int axis) const
- bool [isEnabled](#) () const
- double [keyFactor](#) () const
- double [mouseFactor](#) () const
- QWidget * [parentWidget](#) ()
- const QWidget * [parentWidget](#) () const
- const [QwtPlot * plot](#) () const
- [QwtPlot * plot](#) ()
- [QwtPlotMagnifier](#) ([QwtPlotCanvas *](#))
- void [setAxisEnabled](#) (int axis, bool on)
- void [setEnabled](#) (bool)
- void [setKeyFactor](#) (double)
- void [setMouseButton](#) (int button, int buttonState=[Qt::NoButton](#))
- void [setMouseFactor](#) (double)
- void [setWheelButtonState](#) (int buttonState)
- void [setWheelFactor](#) (double)
- void [setZoomInKey](#) (int key, int modifiers)
- void [setZoomOutKey](#) (int key, int modifiers)
- int [wheelButtonState](#) () const
- double [wheelFactor](#) () const
- virtual [~QwtPlotMagnifier](#) ()

Protected Member Functions

- virtual void [rescale](#) (double factor)
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)
- virtual void [widgetWheelEvent](#) (QWheelEvent *)

12.56.1 Detailed Description

[QwtPlotMagnifier](#) 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 [QwtPlotZoomer](#) and [QwtPlotPanner](#) it is possible to implement individual and powerful navigation of the plot canvas.

See also

[QwtPlotZoomer](#), [QwtPlotPanner](#), [QwtPlot](#)

12.56.2 Constructor & Destructor Documentation

12.56.2.1 QwtPlotMagnifier::QwtPlotMagnifier (QwtPlotCanvas * *canvas*) [explicit]

Constructor

Parameters

<i>canvas</i>	Plot canvas to be magnified
---------------	-----------------------------

12.56.2.2 QwtPlotMagnifier::~QwtPlotMagnifier () [virtual]

Destructor.

12.56.3 Member Function Documentation

12.56.3.1 QwtPlotCanvas * QwtPlotMagnifier::canvas ()

Return observed plot canvas.

12.56.3.2 const QwtPlotCanvas * QwtPlotMagnifier::canvas () const

Return Observed plot canvas.

12.56.3.3 bool QwtMagnifier::eventFilter (QObject * *o*, QEvent * *e*) [virtual, inherited]

Event filter.

When [isEnabled\(\)](#) the mouse events of the observed widget are filtered.

See also

[widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#) [widgetKeyReleaseEvent\(\)](#)

12.56.3.4 `void QwtMagnifier::getMouseButton (int & button, int & buttonState) const` `[inherited]`

See also

[setMouseButton\(\)](#)

12.56.3.5 `void QwtMagnifier::getZoomInKey (int & key, int & modifiers) const` `[inherited]`

See also

[setZoomInKey\(\)](#)

12.56.3.6 `void QwtMagnifier::getZoomOutKey (int & key, int & modifiers) const` `[inherited]`

See also

[setZoomOutKey\(\)](#)

12.56.3.7 `bool QwtPlotMagnifier::isAxisEnabled (int axis) const`

Test if an axis is enabled

Parameters

<i>axis</i>	Axis, see QwtPlot::Axis
-------------	---

Returns

True, if the axis is enabled

See also

[setAxisEnabled\(\)](#)

12.56.3.8 bool QwtMagnifier::isEnabled () const [inherited]

Returns

true when enabled, false otherwise

See also

[setEnabled\(\)](#), [eventFilter\(\)](#)

12.56.3.9 double QwtMagnifier::keyFactor () const [inherited]

Returns

Key factor

See also

[setKeyFactor\(\)](#)

12.56.3.10 double QwtMagnifier::mouseFactor () const [inherited]

Returns

Mouse factor

See also

[setMouseFactor\(\)](#)

12.56.3.11 QWidget * QwtMagnifier::parentWidget () [inherited]

Returns

Parent widget, where the rescaling happens

12.56.3.12 `const QWidget * QwtMagnifier::parentWidget () const`
[inherited]

Returns

Parent widget, where the rescaling happens

12.56.3.13 `QwtPlot * QwtPlotMagnifier::plot ()`

Return plot widget, containing the observed plot canvas.

12.56.3.14 `const QwtPlot * QwtPlotMagnifier::plot () const`

Return plot widget, containing the observed plot canvas.

12.56.3.15 `void QwtPlotMagnifier::rescale (double factor)` **[protected, virtual]**

Zoom in/out the axes scales

Parameters

<i>factor</i>	A value < 1.0 zooms in, a value > 1.0 zooms out.
---------------	--

Implements [QwtMagnifier](#).

12.56.3.16 `void QwtPlotMagnifier::setAxisEnabled (int axis, bool on)`

En/Disable an axis.

Axes that are enabled will be synchronized to the result of panning. All other axes will remain unchanged.

Parameters

<i>axis</i>	Axis, see QwtPlot::Axis
<i>on</i>	On/Off

See also

[isAxisEnabled\(\)](#)

12.56.3.17 void QwtMagnifier::setEnabled (bool *on*) [inherited]

En/disable the magnifier.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>on</i>	true or false
-----------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.56.3.18 void QwtMagnifier::setKeyFactor (double *factor*) [inherited]

Change the key factor.

The key factor defines the ratio between the current range on the parent widget and the zoomed range for each key press of the zoom in/out keys. The default value is 0.9.

Parameters

<i>factor</i>	Key factor
---------------	------------

See also

[keyFactor\(\)](#), [setZoomInKey\(\)](#), [setZoomOutKey\(\)](#), [setWheelFactor](#), [setMouseFactor\(\)](#)

12.56.3.19 void QwtMagnifier::setMouseButton (int *button*, int *buttonState* = *Qt::NoButton*) [inherited]

Assign the mouse button, that is used for zooming in/out. The default value is `Qt::RightButton`.

Parameters

<i>button</i>	Button
<i>buttonState</i>	Button state

See also

[getMouseButton\(\)](#)

12.56.3.20 void QwtMagnifier::setMouseFactor (double *factor*) [inherited]

Change the mouse factor.

The mouse factor defines the ratio between the current range on the parent widget and the zoomed range for each vertical mouse movement. The default value is 0.95.

Parameters

<i>factor</i>	Wheel factor
---------------	--------------

See also

[mouseFactor\(\)](#), [setMouseButton\(\)](#), [setWheelFactor\(\)](#), [setKeyFactor\(\)](#)

12.56.3.21 void QwtMagnifier::setWheelButtonState (int *buttonState*) [inherited]

Assign a mandatory button state for zooming in/out using the wheel. The default button state is Qt::NoButton.

Parameters

<i>buttonState</i>	Button state
--------------------	--------------

See also

[wheelButtonState\(\)](#)

12.56.3.22 void QwtMagnifier::setWheelFactor (double *factor*) [inherited]

Change the wheel factor.

The wheel factor defines the ratio between the current range on the parent widget and the zoomed range for each step of the wheel. The default value is 0.9.

Parameters

<i>factor</i>	Wheel factor
---------------	--------------

See also

[wheelFactor\(\)](#), [setWheelButtonState\(\)](#), [setMouseFactor\(\)](#), [setKeyFactor\(\)](#)

12.56.3.23 void QwtMagnifier::setZoomInKey (int *key*, int *modifiers*)
[inherited]

Assign the key, that is used for zooming in. The default combination is Qt::Key_Plus + Qt::NoModifier.

Parameters

<i>key</i>	
<i>modifiers</i>	

See also

[getZoomInKey\(\)](#), [setZoomOutKey\(\)](#)

12.56.3.24 void QwtMagnifier::setZoomOutKey (int *key*, int *modifiers*)
[inherited]

Assign the key, that is used for zooming out. The default combination is Qt::Key_Minus + Qt::NoModifier.

Parameters

<i>key</i>	
<i>modifiers</i>	

See also

[getZoomOutKey\(\)](#), [setZoomOutKey\(\)](#)

12.56.3.25 int QwtMagnifier::wheelButtonState () const **[inherited]****Returns**

Wheel button state

See also

[setWheelButtonState\(\)](#)

12.56.3.26 double QwtMagnifier::wheelFactor () const **[inherited]****Returns**

Wheel factor

See also

[setWheelFactor\(\)](#)

12.56.3.27 void QwtMagnifier::widgetKeyPressEvent (QKeyEvent * *ke*)
[protected, virtual, inherited]

Handle a key press event for the observed widget.

Parameters

<i>ke</i>	Key event
-----------	-----------

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.56.3.28 void QwtMagnifier::widgetKeyReleaseEvent (QKeyEvent *)
[protected, virtual, inherited]

Handle a key release event for the observed widget.

Parameters

<i>ke</i>	Key event
-----------	-----------

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.56.3.29 void QwtMagnifier::widgetMouseMoveEvent (QMouseEvent * *me*)
[protected, virtual, inherited]

Handle a mouse move event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#),

12.56.3.30 void QwtMagnifier::widgetMousePressEvent (QMouseEvent * *me*)
[protected, virtual, inherited]

Handle a mouse press event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMouseEventReleaseEvent\(\)](#), [widgetMouseEventMoveEvent\(\)](#)

12.56.3.31 `void QwtMagnifier::widgetMouseEvent (QMouseEvent * me)`
[protected, virtual, inherited]

Handle a mouse release event for the observed widget.

See also

[eventFilter\(\)](#), [widgetMouseEventPressEvent\(\)](#), [widgetMouseEventMoveEvent\(\)](#),

12.56.3.32 `void QwtMagnifier::widgetWheelEvent (QWheelEvent * we)`
[protected, virtual, inherited]

Handle a wheel event for the observed widget.

Parameters

<i>we</i>	Wheel event
-----------	-------------

See also

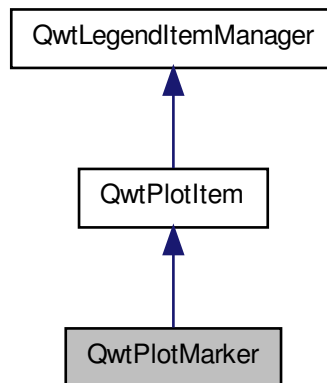
[eventFilter\(\)](#)

12.57 QwtPlotMarker Class Reference

A class for drawing markers.

```
#include <qwt_plot_marker.h>
```

Inheritance diagram for QwtPlotMarker:



Public Types

- enum [ItemAttribute](#) {
 Legend = 1,
 AutoScale = 2 }
- enum [LineStyle](#) {
 NoLine,
 HLine,
 VLine,
 Cross }
- enum [RenderHint](#) { **RenderAntialiased** = 1 }
- enum [RttiValues](#) {
 Rtti_PlotItem = 0,
 Rtti_PlotGrid,
 Rtti_PlotScale,
 Rtti_PlotMarker,
 Rtti_PlotCurve,
 Rtti_PlotHistogram,
 Rtti_PlotSpectrogram,
 Rtti_PlotSVG,
 Rtti_PlotUserItem = 1000 }

Public Member Functions

- void [attach](#) (QwtPlot *plot)
- virtual QwtDoubleRect [boundingRect](#) () const
- void [detach](#) ()
- virtual void [draw](#) (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &) const
- void [hide](#) ()
- QwtDoubleRect [invTransform](#) (const QwtScaleMap &, const QwtScaleMap &, const QRect &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- [QwtText](#) [label](#) () const
- Qt::Alignment [labelAlignment](#) () const
- Qt::Orientation [labelOrientation](#) () const
- virtual QWidget * [legendItem](#) () const
- const QPen & [linePen](#) () const
- [LineStyle](#) [lineStyle](#) () const
- QRect [paintRect](#) (const QwtScaleMap &, const QwtScaleMap &) const
- QwtPlot * [plot](#) () const
- [QwtPlotMarker](#) ()
- virtual int [rtti](#) () const
- QwtDoubleRect [scaleRect](#) (const QwtScaleMap &, const QwtScaleMap &) const
- void [setAxis](#) (int xAxis, int yAxis)
- void [setItemAttribute](#) (ItemAttribute, bool on=true)
- void [setLabel](#) (const QwtText &)
- void [setLabelAlignment](#) (Qt::Alignment)
- void [setLabelOrientation](#) (Qt::Orientation)
- void [setLinePen](#) (const QPen &p)
- void [setLineStyle](#) (LineStyle st)
- void [setRenderHint](#) (RenderHint, bool on=true)
- void [setSpacing](#) (int)
- void [setSymbol](#) (const QwtSymbol &s)
- void [setTitle](#) (const QString &title)
- void [setTitle](#) (const QwtText &title)
- void [setValue](#) (double, double)
- void [setValue](#) (const QwtDoublePoint &)
- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setXValue](#) (double)
- void [setYAxis](#) (int axis)
- void [setYValue](#) (double)
- void [setZ](#) (double z)
- void [show](#) ()
- int [spacing](#) () const
- const QwtSymbol & [symbol](#) () const
- bool [testItemAttribute](#) (ItemAttribute) const

- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- [QRect](#) [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QwtDoubleRect](#) &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- [QwtDoublePoint](#) [value](#) () const
- int [xAxis](#) () const
- double [xValue](#) () const
- int [yAxis](#) () const
- double [yValue](#) () const
- double [z](#) () const
- virtual [~QwtPlotMarker](#) ()

Protected Member Functions

- void [drawAt](#) ([QPainter](#) *, const [QRect](#) &, const [QPoint](#) &) const

12.57.1 Detailed Description

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

The [QwtPlotMarker::setSymbol\(\)](#) member assigns a symbol to the marker. The symbol is drawn at the specified point.

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

12.57.2 Member Enumeration Documentation

12.57.2.1 enum [QwtPlotItem::ItemAttribute](#) [inherited]

Plot Item Attributes

- Legend
The item is represented on the legend.
- AutoScale
The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.57.2.2 enum QwtPlotMarker::LineStyle

Line styles.

See also

[setLineStyle\(\)](#), [lineStyle\(\)](#)

12.57.2.3 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.57.2.4 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.57.3 Constructor & Destructor Documentation

12.57.3.1 QwtPlotMarker::QwtPlotMarker () [explicit]

Sets alignment to Qt::AlignCenter, and style to NoLine.

12.57.3.2 QwtPlotMarker::~QwtPlotMarker () [virtual]

Destructor.

12.57.4 Member Function Documentation

12.57.4.1 void QwtPlotItem::attach (QwtPlot *plot) [inherited]

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.57.4.2 `QwtDoubleRect QwtPlotMarker::boundingRect () const`
[virtual]

Returns

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

Reimplemented from [QwtPlotItem](#).

12.57.4.3 `void QwtPlotItem::detach () [inline, inherited]`

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with.
[detach\(\)](#) is equivalent to calling `attach(NULL)`

See also

[attach\(QwtPlot* plot \)](#)

12.57.4.4 `void QwtPlotMarker::draw (QPainter * painter, const
QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect &
canvasRect) const [virtual]`

Draw the marker

Parameters

<i>painter</i>	Painter
<i>xMap</i>	x Scale Map
<i>yMap</i>	y Scale Map
<i>canvasRect</i>	Contents rect of the canvas in painter coordinates

Implements [QwtPlotItem](#).

12.57.4.5 `void QwtPlotMarker::drawAt (QPainter * painter, const QRect & canvasRect, const QPoint & pos) const` **[protected]**

Draw the marker at a specific position

Parameters

<i>painter</i>	Painter
<i>canvasRect</i>	Contents rect of the canvas in painter coordinates
<i>pos</i>	Position of the marker in painter coordinates

12.57.4.6 `void QwtPlotItem::hide ()` **[inherited]**

Hide the item.

12.57.4.7 `QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & rect) const` **[inherited]**

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.57.4.8 `bool QwtPlotItem::isVisible () const` **[inherited]**

Returns

true if visible

See also

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

12.57.4.9 void QwtPlotItem::itemChanged () [virtual, inherited]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.57.4.10 QwtText QwtPlotMarker::label () const

Returns

the label

See also

[setLabel\(\)](#)

12.57.4.11 Qt::Alignment QwtPlotMarker::labelAlignment () const

Returns

the label alignment

See also

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

12.57.4.12 Qt::Orientation QwtPlotMarker::labelOrientation () const

Returns

the label orientation

See also

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

12.57.4.13 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.57.4.14 const QPen & QwtPlotMarker::linePen () const**Returns**

the line pen

See also

[setLinePen\(\)](#)

12.57.4.15 QwtPlotMarker::LineStyle QwtPlotMarker::lineStyle () const**Returns**

the line style

See also

For a description of line styles, see [QwtPlotMarker::setLineStyle\(\)](#)

12.57.4.16 QRect QwtPlotItem::paintRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.57.4.17 QwtPlot * QwtPlotItem::plot () const [inherited]

Return attached plot.

12.57.4.18 int QwtPlotMarker::rtti () const [virtual]**Returns**

QwtPlotItem::Rtti_PlotMarker

Reimplemented from [QwtPlotItem](#).

12.57.4.19 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.57.4.20 void QwtPlotItem::setAxis (int xAxis, int yAxis) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.57.4.21 `void QwtPlotItem::setItemAttribute (ItemAttribute attribute, bool on = true) [inherited]`

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.57.4.22 `void QwtPlotMarker::setLabel (const QwtText & label)`

Set the label.

Parameters

<i>label</i>	label text
--------------	------------

See also

[label\(\)](#)

12.57.4.23 `void QwtPlotMarker::setLabelAlignment (Qt::Alignment align)`

Set the alignment of the label.

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

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

Parameters

<i>align</i>	Alignment. A combination of <code>AlignTop</code> , <code>AlignBottom</code> , <code>AlignLeft</code> , <code>AlignRight</code> , <code>AlignCenter</code> , <code>AlignHCenter</code> , <code>AlignVCenter</code> .
--------------	--

See also

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

12.57.4.24 void QwtPlotMarker::setLabelOrientation (Qt::Orientation *orientation*)

Set the orientation of the label.

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

Parameters

<i>orientation</i>	Orientation of the label
--------------------	--------------------------

See also

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

12.57.4.25 void QwtPlotMarker::setLinePen (const QPen & *pen*)

Specify a pen for the line.

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

<i>pen</i>	New pen
------------	---------

See also

[linePen\(\)](#), [QwtPainter::scaledPen\(\)](#)

12.57.4.26 void QwtPlotMarker::setLineStyle (QwtPlotMarker::LineStyle *st*)

Set the line style.

Parameters

<i>st</i>	Line style. Can be one of QwtPlotMarker::NoLine, HLine, VLine or Cross
-----------	--

See also

[lineStyle\(\)](#)

12.57.4.27 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on* = *true*) [inherited]

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.57.4.28 void QwtPlotMarker::setSpacing (int *spacing*)

Set the spacing.

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

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

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

12.57.4.29 void QwtPlotMarker::setSymbol (const QwtSymbol & *s*)

Assign a symbol.

Parameters

<i>s</i>	New symbol
----------	------------

See also

[symbol\(\)](#)

12.57.4.30 void QwtPlotItem::setTitle (const QString & *title*) [inherited]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also[title\(\)](#)

12.57.4.31 void QwtPlotItem::setTitle (const QwtText & *title*)
[**inherited**]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also[title\(\)](#)

12.57.4.32 void QwtPlotMarker::setValue (const QwtDoublePoint & *pos*)

Set Value.

12.57.4.33 void QwtPlotMarker::setValue (double *x*, double *y*)

Set Value.

12.57.4.34 void QwtPlotItem::setVisible (bool *on*) [**virtual**,
inherited]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also[isVisible\(\)](#), [show\(\)](#), [hide\(\)](#)

12.57.4.35 void QwtPlotItem::setXAxis (int *axis*) [**inherited**]

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also

[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)

12.57.4.36 void QwtPlotMarker::setXValue (double *x*)

Set X Value.

12.57.4.37 void QwtPlotItem::setYAxis (int *axis*) [inherited]

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also

[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)

12.57.4.38 void QwtPlotMarker::setYValue (double *y*)

Set Y Value.

12.57.4.39 void QwtPlotItem::setZ (double *z*) [inherited]

Set the *z* value.

Plot items are painted in increasing *z*-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also

[z\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.57.4.40 void QwtPlotItem::show () [inherited]

Show the item.

12.57.4.41 int QwtPlotMarker::spacing () const**Returns**

the spacing

See also

[setSpacing\(\)](#)

12.57.4.42 const QwtSymbol & QwtPlotMarker::symbol () const**Returns**

the symbol

See also

[setSymbol\(\)](#), [QwtSymbol](#)

**12.57.4.43 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*)
const [inherited]**

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also

[setItemAttribute\(\)](#), [ItemAttribute](#)

12.57.4.44 `bool QwtPlotItem::testRenderHint (RenderHint hint) const`
[[inherited](#)]

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.57.4.45 `const QwtText & QwtPlotItem::title () const` [[inherited](#)]

Returns

Title of the item

See also

[setTitle\(\)](#)

12.57.4.46 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap,
const QwtScaleMap & yMap, const QwtDoubleRect & rect) const`
[[inherited](#)]

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.57.4.47 void QwtPlotItem::updateLegend (QwtLegend * *legend*) const [virtual, inherited]

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also

[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

12.57.4.48 void QwtPlotItem::updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &) [virtual, inherited]

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 [QwtPlotGrid\(\)](#)) have to reimplement [updateScaleDiv\(\)](#)

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented in [QwtPlotGrid](#), and [QwtPlotScaleItem](#).

12.57.4.49 QwtDoublePoint QwtPlotMarker::value () const

Return Value.

12.57.4.50 int QwtPlotItem::xAxis () const [inherited]

Return xAxis.

12.57.4.51 double QwtPlotMarker::xValue () const

Return x Value.

12.57.4.52 int QwtPlotItem::yAxis () const [inherited]

Return yAxis.

12.57.4.53 double QwtPlotMarker::yValue () const

Return y Value.

12.57.4.54 double QwtPlotItem::z () const [inherited]

Plot items are painted in increasing z-order.

Returns

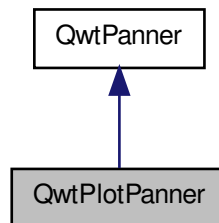
[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.58 QwtPlotPanner Class Reference

[QwtPlotPanner](#) provides panning of a plot canvas.

```
#include <qwt_plot_panner.h>
```

Inheritance diagram for QwtPlotPanner:



Signals

- void [moved](#) (int dx, int dy)
- void [panned](#) (int dx, int dy)

Public Member Functions

- [QwtPlotCanvas](#) * [canvas](#) ()
- const [QwtPlotCanvas](#) * [canvas](#) () const
- const [QCursor](#) [cursor](#) () const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [getAbortKey](#) (int &key, int &state) const
- void [getMouseButton](#) (int &button, int &buttonState) const
- bool [isAxisEnabled](#) (int axis) const
- bool [isEnabled](#) () const
- bool [isOrientationEnabled](#) (Qt::Orientation) const
- Qt::Orientations [orientations](#) () const
- [QwtPlot](#) * [plot](#) ()
- const [QwtPlot](#) * [plot](#) () const
- [QwtPlotPanner](#) ([QwtPlotCanvas](#) *)
- void [setAbortKey](#) (int key, int state=Qt::NoButton)
- void [setAxisEnabled](#) (int axis, bool on)
- void [setCursor](#) (const [QCursor](#) &)
- void [setEnabled](#) (bool)
- void [setMouseButton](#) (int button, int buttonState=Qt::NoButton)
- void [setOrientations](#) (Qt::Orientations)
- virtual ~[QwtPlotPanner](#) ()

Protected Slots

- virtual void [moveCanvas](#) (int dx, int dy)

Protected Member Functions

- virtual void [paintEvent](#) (QPaintEvent *)
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)

12.58.1 Detailed Description

[QwtPlotPanner](#) provides panning of a plot canvas. [QwtPlotPanner](#) is a panner for a [QwtPlotCanvas](#), that adjusts the scales of the axes after dropping the canvas on its new position.

Together with [QwtPlotZoomer](#) and [QwtPlotMagnifier](#) powerful ways of navigating on a [QwtPlot](#) widget can be implemented easily.

Note

The axes are not updated, while dragging the canvas

See also

[QwtPlotZoomer](#), [QwtPlotMagnifier](#)

12.58.2 Constructor & Destructor Documentation**12.58.2.1 QwtPlotPanner::QwtPlotPanner ([QwtPlotCanvas](#) * *canvas*)
[explicit]**

Create a plot panner.

The panner is enabled for all axes

Parameters

<i>canvas</i>	Plot canvas to pan, also the parent object
---------------	--

See also

[setAxisEnabled\(\)](#)

12.58.2.2 QwtPlotPanner::~~QwtPlotPanner () [virtual]

Destructor.

12.58.3 Member Function Documentation

12.58.3.1 QwtPlotCanvas * QwtPlotPanner::canvas ()

Return observed plot canvas.

12.58.3.2 const QwtPlotCanvas * QwtPlotPanner::canvas () const

Return Observed plot canvas.

12.58.3.3 const QCursor QwtPanner::cursor () const [inherited]

Returns

Cursor that is active while panning

See also

[setCursor\(\)](#)

12.58.3.4 bool QwtPanner::eventFilter (QObject * o, QEvent * e) [virtual, inherited]

Event filter.

When [isEnabled\(\)](#) the mouse events of the observed widget are filtered.

See also

[widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseMoveEvent\(\)](#)

12.58.3.5 void QwtPanner::getAbortKey (int & key, int & state) const [inherited]

Get the abort key.

12.58.3.6 void QwtPanner::getMouseButton (int & *button*, int & *buttonState*)
const [inherited]

Get the mouse button.

12.58.3.7 bool QwtPlotPanner::isAxisEnabled (int *axis*) const

Test if an axis is enabled

Parameters

<i>axis</i> Axis, see QwtPlot::Axis

Returns

True, if the axis is enabled

See also

[setAxisEnabled\(\)](#), [moveCanvas\(\)](#)

12.58.3.8 bool QwtPanner::isEnabled () const [inherited]

Returns

true when enabled, false otherwise

See also

[setEnabled](#), [eventFilter\(\)](#)

12.58.3.9 bool QwtPanner::isOrientationEnabled (Qt::Orientation *o*) const
[inherited]

Return true if a orientatio is enabled

See also

[orientations\(\)](#), [setOrientations\(\)](#)

12.58.3.10 void QwtPlotPanner::moveCanvas (int *dx*, int *dy*)
[protected, virtual, slot]

Adjust the enabled axes according to dx/dy

Parameters

<i>dx</i>	Pixel offset in x direction
<i>dy</i>	Pixel offset in y direction

See also

[QwtPanner::panned\(\)](#)

12.58.3.11 `void QwtPanner::moved (int dx, int dy) [signal, inherited]`

Signal emitted, while the widget moved, but panning is not finished.

Parameters

<i>dx</i>	Offset in horizontal direction
<i>dy</i>	Offset in vertical direction

12.58.3.12 `Qt::Orientations QwtPanner::orientations () const [inherited]`

Return the orientation, where panning is enabled.

12.58.3.13 `void QwtPanner::paintEvent (QPaintEvent * pe) [protected, virtual, inherited]`

Paint event.

Repaint the grabbed pixmap on its current position and fill the empty spaces by the background of the parent widget.

Parameters

<i>pe</i>	Paint event
-----------	-------------

12.58.3.14 `void QwtPanner::panned (int dx, int dy) [signal, inherited]`

Signal emitted, when panning is done

Parameters

<i>dx</i>	Offset in horizontal direction
<i>dy</i>	Offset in vertical direction

12.58.3.15 `const QwtPlot * QwtPlotPanner::plot () const`

Return plot widget, containing the observed plot canvas.

12.58.3.16 `QwtPlot * QwtPlotPanner::plot ()`

Return plot widget, containing the observed plot canvas.

12.58.3.17 `void QwtPanner::setAbortKey (int key, int state = Qt::NoButton) [inherited]`

Change the abort key The defaults are Qt::Key_Escape and Qt::NoButton

Parameters

<i>key</i>	Key (See Qt::Keycode)
<i>state</i>	State

12.58.3.18 `void QwtPlotPanner::setAxisEnabled (int axis, bool on)`

En/Disable an axis.

Axes that are enabled will be synchronized to the result of panning. All other axes will remain unchanged.

Parameters

<i>axis</i>	Axis, see QwtPlot::Axis
<i>on</i>	On/Off

See also

[isAxisEnabled\(\)](#), [moveCanvas\(\)](#)

12.58.3.19 `void QwtPanner::setCursor (const QCursor & cursor) [inherited]`

Change the cursor, that is active while panning The default is the cursor of the parent widget.

Parameters

<i>cursor</i>	New cursor
---------------	------------

See also

[setCursor\(\)](#)

12.58.3.20 void QwtPanner::setEnabled (bool *on*) [inherited]

En/disable the panner.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>on</i>	true or false
-----------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.58.3.21 void QwtPanner::setMouseButton (int *button*, int *buttonState* = *Qt::NoButton*) [inherited]

Change the mouse button The defaults are Qt::LeftButton and Qt::NoButton

12.58.3.22 void QwtPanner::setOrientations (Qt::Orientations *o*) [inherited]

Set the orientations, where panning is enabled The default value is in both directions: Qt::Horizontal | Qt::Vertical

/param o Orientation

12.58.3.23 void QwtPanner::widgetKeyPressEvent (QKeyEvent * *ke*) [protected, virtual, inherited]

Handle a key press event for the observed widget.

Parameters

<i>ke</i>	Key event
-----------	-----------

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.58.3.24 void QwtPanner::widgetKeyReleaseEvent (QKeyEvent * *me*)
[protected, virtual, inherited]

Handle a key release event for the observed widget.

See also

[eventFilter\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.58.3.25 void QwtPanner::widgetMouseMoveEvent (QMouseEvent * *me*)
[protected, virtual, inherited]

Handle a mouse move event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#)

12.58.3.26 void QwtPanner::widgetMousePressEvent (QMouseEvent * *me*)
[protected, virtual, inherited]

Handle a mouse press event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

[eventFilter\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#),

12.58.3.27 void QwtPanner::widgetMouseReleaseEvent (QMouseEvent * *me*)
[protected, virtual, inherited]

Handle a mouse release event for the observed widget.

Parameters

<i>me</i>	Mouse event
-----------	-------------

See also

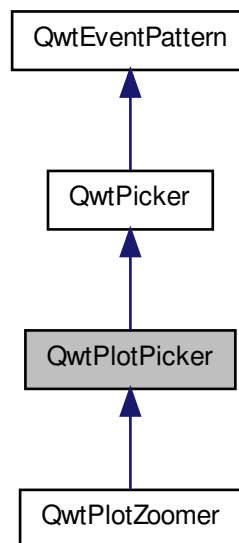
[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseMoveEvent\(\)](#),

12.59 QwtPlotPicker Class Reference

[QwtPlotPicker](#) provides selections on a plot canvas.

```
#include <qwt_plot_picker.h>
```

Inheritance diagram for QwtPlotPicker:



Public Types

- enum [DisplayMode](#) {
 AlwaysOff,
 AlwaysOn,
 ActiveOnly }
- enum [KeyPatternCode](#) {
 KeySelect1,
 KeySelect2,
 KeyAbort,
 KeyLeft,
 KeyRight,
 KeyUp,

```
    KeyDown,  
    KeyRedo,  
    KeyUndo,  
    KeyHome,  
    KeyPatternCount }  
• enum MousePatternCode {  
    MouseSelect1,  
    MouseSelect2,  
    MouseSelect3,  
    MouseSelect4,  
    MouseSelect5,  
    MouseSelect6,  
    MousePatternCount }  
• enum RectSelectionType {  
    CornerToCorner = 64,  
    CenterToCorner = 128,  
    CenterToRadius = 256 }  
• enum ResizeMode {  
    Stretch,  
    KeepSize }  
• enum RubberBand {  
    NoRubberBand = 0,  
    HLineRubberBand,  
    VLineRubberBand,  
    CrossRubberBand,  
    RectRubberBand,  
    EllipseRubberBand,  
    PolygonRubberBand,  
    UserRubberBand = 100 }  
• enum SelectionMode {  
    ClickSelection = 1024,  
    DragSelection = 2048 }  
• enum SelectionType {  
    NoSelection = 0,  
    PointSelection = 1,  
    RectSelection = 2,  
    PolygonSelection = 4 }
```

Signals

- void [appended](#) (const QwtDoublePoint &pos)
- void [appended](#) (const QPoint &pos)
- void [changed](#) (const QwtPolygon &pa)
- void [moved](#) (const QwtDoublePoint &pos)
- void [moved](#) (const QPoint &pos)
- void [selected](#) (const QwtDoublePoint &pos)
- void [selected](#) (const QwtDoubleRect &rect)
- void [selected](#) (const QwtPolygon &pa)
- void [selected](#) (const QwtArray< QwtDoublePoint > &pa)

Public Member Functions

- [QwtPlotCanvas](#) * [canvas](#) ()
- const [QwtPlotCanvas](#) * [canvas](#) () const
- virtual void [drawRubberBand](#) (QPainter *) const
- virtual void [drawTracker](#) (QPainter *) const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [initKeyPattern](#) ()
- void [initMousePattern](#) (int numButtons)
- bool [isActive](#) () const
- bool [isEnabled](#) () const
- bool [keyMatch](#) (uint pattern, const QKeyEvent *) const
- const QwtArray< [KeyPattern](#) > & [keyPattern](#) () const
- QwtArray< [KeyPattern](#) > & [keyPattern](#) ()
- bool [mouseMatch](#) (uint pattern, const QMouseEvent *) const
- QwtArray< [MousePattern](#) > & [mousePattern](#) ()
- const QwtArray< [MousePattern](#) > & [mousePattern](#) () const
- QWidget * [parentWidget](#) ()
- const QWidget * [parentWidget](#) () const
- virtual QRect [pickRect](#) () const
- const [QwtPlot](#) * [plot](#) () const
- [QwtPlot](#) * [plot](#) ()
- [QwtPlotPicker](#) ([QwtPlotCanvas](#) *)
- [QwtPlotPicker](#) (int xAxis, int yAxis, int selectionFlags, [RubberBand](#) rubberBand, [DisplayMode](#) trackerMode, [QwtPlotCanvas](#) *)
- [QwtPlotPicker](#) (int xAxis, int yAxis, [QwtPlotCanvas](#) *)
- [ResizeMode](#) [resizeMode](#) () const
- [RubberBand](#) [rubberBand](#) () const
- QPen [rubberBandPen](#) () const
- const QwtPolygon & [selection](#) () const
- int [selectionFlags](#) () const
- virtual void [setAxis](#) (int xAxis, int yAxis)
- virtual void [setEnabled](#) (bool)
- void [setKeyPattern](#) (uint pattern, int key, int state=Qt::NoButton)

- void [setKeyPattern](#) (const QwtArray< [KeyPattern](#) > &)
- void [setMousePattern](#) (const QwtArray< [MousePattern](#) > &)
- void [setMousePattern](#) (uint pattern, int button, int state=Qt::NoButton)
- virtual void [setResizeMode](#) ([ResizeMode](#))
- virtual void [setRubberBand](#) ([RubberBand](#))
- virtual void [setRubberBandPen](#) (const QPen &)
- virtual void [setSelectionFlags](#) (int)
- virtual void [setTrackerFont](#) (const QFont &)
- virtual void [setTrackerMode](#) ([DisplayMode](#))
- virtual void [setTrackerPen](#) (const QPen &)
- QFont [trackerFont](#) () const
- [DisplayMode](#) [trackerMode](#) () const
- QPen [trackerPen](#) () const
- QPoint [trackerPosition](#) () const
- QRect [trackerRect](#) (const QFont &) const
- int [xAxis](#) () const
- int [yAxis](#) () const
- virtual ~[QwtPlotPicker](#) ()

Protected Member Functions

- virtual bool [accept](#) (QwtPolygon &selection) const
- virtual void [append](#) (const QPoint &)
- virtual void [begin](#) ()
- virtual bool [end](#) (bool ok=true)
- QwtDoublePoint [invTransform](#) (const QPoint &) const
- QwtDoubleRect [invTransform](#) (const QRect &) const
- virtual bool [keyMatch](#) (const [KeyPattern](#) &, const QKeyEvent *) const
- virtual bool [mouseMatch](#) (const [MousePattern](#) &, const QMouseEvent *) const
- virtual void [move](#) (const QPoint &)
- virtual void [reset](#) ()
- const QWidget * [rubberBandWidget](#) () const
- QwtDoubleRect [scaleRect](#) () const
- virtual [QwtPickerMachine](#) * [stateMachine](#) (int) const
- virtual void [stretchSelection](#) (const QSize &oldSize, const QSize &newSize)
- virtual [QwtText](#) [trackerText](#) (const QPoint &) const
- virtual [QwtText](#) [trackerText](#) (const QwtDoublePoint &) const
- const QWidget * [trackerWidget](#) () const
- QPoint [transform](#) (const QwtDoublePoint &) const
- QRect [transform](#) (const QwtDoubleRect &) const
- virtual void [transition](#) (const QEvent *)
- virtual void [updateDisplay](#) ()
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetLeaveEvent](#) (QEvent *)
- virtual void [widgetMouseDoubleClickEvent](#) (QMouseEvent *)

- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)
- virtual void [widgetWheelEvent](#) (QWheelEvent *)

12.59.1 Detailed Description

[QwtPlotPicker](#) provides selections on a plot canvas. [QwtPlotPicker](#) is a [QwtPicker](#) tailored for selections on a plot canvas. It is set to a x-Axis and y-Axis and translates all pixel coordinates into this coordinate system.

12.59.2 Member Enumeration Documentation

12.59.2.1 enum QwtPicker::DisplayMode [inherited]

- AlwaysOff
Display never.
- AlwaysOn
Display always.
- ActiveOnly
Display only when the selection is active.

See also

[QwtPicker::setTrackerMode\(\)](#), [QwtPicker::trackerMode\(\)](#), [QwtPicker::isActive\(\)](#)

12.59.2.2 enum QwtEventPattern::KeyPatternCode [inherited]

Symbolic keyboard input codes.

Default initialization:

- KeySelect1
Qt::Key_Return
- KeySelect2
Qt::Key_Space
- KeyAbort
Qt::Key_Escape

- KeyLeft
Qt::Key_Left
- KeyRight
Qt::Key_Right
- KeyUp
Qt::Key_Up
- KeyDown
Qt::Key_Down
- KeyUndo
Qt::Key_Minus
- KeyRedo
Qt::Key_Plus
- KeyHome
Qt::Key_Escape

12.59.2.3 enum QwtEventPattern::MousePatternCode [inherited]

Symbolic mouse input codes.

The default initialization for 3 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::MidButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::MidButton + Qt::ShiftButton

The default initialization for 2 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

The default initialization for 1 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::LeftButton + Qt::ControlButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::LeftButton + Qt::ControlButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

See also

[initMousePattern\(\)](#)

12.59.2.4 enum QwtPicker::RectSelectionType [inherited]

Selection subtype for RectSelection This enum type describes the type of rectangle selections. It can be or'd with [QwtPicker::RectSelectionType](#) and [QwtPicker::SelectionMode](#) and passed to [QwtPicker::setSelectionFlags\(\)](#).

- CornerToCorner
The first and the second selected point are the corners of the rectangle.
- CenterToCorner
The first point is the center, the second a corner of the rectangle.
- CenterToRadius
The first point is the center of a quadrat, calculated by the maximum of the x- and y-distance.

The default value is CornerToCorner.

See also

[QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::selectionFlags\(\)](#)

12.59.2.5 enum QwtPicker::ResizeMode [inherited]

Controls what to do with the selected points of an active selection when the observed widget is resized.

- Stretch
All points are scaled according to the new size,
- KeepSize
All points remain unchanged.

The default value is Stretch.

See also

[QwtPicker::setResizeMode\(\)](#), [QwtPicker::resize\(\)](#)

12.59.2.6 enum QwtPicker::RubberBand [inherited]

Rubberband style

- NoRubberBand
No rubberband.

- `HLineRubberBand` & `PointSelection`
A horizontal line.
- `VLineRubberBand` & `PointSelection`
A vertical line.
- `CrossRubberBand` & `PointSelection`
A horizontal and a vertical line.
- `RectRubberBand` & `RectSelection`
A rectangle.
- `EllipseRubberBand` & `RectSelection`
An ellipse.
- `PolygonRubberBand` & `PolygonSelection`
A polygon.
- `UserRubberBand`
Values \geq `UserRubberBand` can be used to define additional rubber bands.

The default value is `NoRubberBand`.

See also

[QwtPicker::setRubberBand\(\)](#), [QwtPicker::rubberBand\(\)](#)

12.59.2.7 enum QwtPicker::SelectionMode [inherited]

Values of this enum type or'd together with a `SelectionType` value identifies which state machine should be used for the selection.

The default value is `ClickSelection`.

See also

[stateMachine\(\)](#)

12.59.2.8 enum QwtPicker::SelectionType [inherited]

This enum type describes the type of a selection. It can be or'd with [QwtPicker::RectSelectionType](#) and [QwtPicker::SelectionMode](#) and passed to [QwtPicker::setSelectionFlags\(\)](#)

- `NoSelection`
Selection is disabled. Note this is different to the disabled state, as you might have a tracker.

- **PointSelection**
Select a single point.
- **RectSelection**
Select a rectangle.
- **PolygonSelection**
Select a polygon.

The default value is NoSelection.

See also

[QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::selectionFlags\(\)](#)

12.59.3 Constructor & Destructor Documentation

12.59.3.1 QwtPlotPicker::QwtPlotPicker (QwtPlotCanvas * *canvas*) [explicit]

Create a plot picker.

The picker is set to those x- and y-axis of the plot that are enabled. If both or no x-axis are enabled, the picker is set to QwtPlot::xBottom. If both or no y-axis are enabled, it is set to QwtPlot::yLeft.

Parameters

<i>canvas</i>	Plot canvas to observe, also the parent object
---------------	--

See also

[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#), [QwtPlotPicker::scaleRect\(\)](#)

12.59.3.2 QwtPlotPicker::~QwtPlotPicker () [virtual]

Destructor.

12.59.3.3 QwtPlotPicker::QwtPlotPicker (int *xAxis*, int *yAxis*, QwtPlotCanvas * *canvas*) [explicit]

Create a plot picker

Parameters

<i>xAxis</i>	Set the x axis of the picker
--------------	------------------------------

<i>yAxis</i>	Set the y axis of the picker
<i>canvas</i>	Plot canvas to observe, also the parent object

See also

[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#), [QwtPlotPicker::scaleRect\(\)](#)

12.59.3.4 QwtPlotPicker::QwtPlotPicker (int *xAxis*, int *yAxis*, int *selectionFlags*, RubberBand *rubberBand*, DisplayMode *trackerMode*, QwtPlotCanvas * *canvas*) [explicit]

Create a plot picker

Parameters

<i>xAxis</i>	X axis of the picker
<i>yAxis</i>	Y axis of the picker
<i>selection-Flags</i>	Or'd value of SelectionType, RectSelectionType and SelectionMode
<i>rubberBand</i>	Rubberband style
<i>trackerMode</i>	Tracker mode
<i>canvas</i>	Plot canvas to observe, also the parent object

See also

[QwtPicker](#), [QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::setRubberBand\(\)](#), [QwtPicker::setTrackerMode\(\)](#), [QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#), [QwtPlotPicker::scaleRect\(\)](#)

12.59.4 Member Function Documentation

12.59.4.1 bool QwtPicker::accept (QwtPolygon & *selection*) const
[protected, virtual, inherited]

Validate and fixup the selection.

Accepts all selections unmodified

Parameters

<i>selection</i>	Selection to validate and fixup
------------------	---------------------------------

Returns

true, when accepted, false otherwise

Reimplemented in [QwtPlotZoomer](#).

12.59.4.2 void QwtPlotPicker::append (const QPoint & *pos*) [protected, virtual]

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

Parameters

<i>pos</i>	Additional point
------------	------------------

See also

[isActive](#), [begin\(\)](#), [end\(\)](#), [move\(\)](#), [appended\(\)](#)

Note

The [appended\(const QPoint &\)](#), [appended\(const QDoublePoint &\)](#) signals are emitted.

Reimplemented from [QwtPicker](#).

12.59.4.3 void QwtPlotPicker::appended (const QwtDoublePoint & *pos*) [signal]

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

Parameters

<i>pos</i>	Position of the appended point.
------------	---------------------------------

See also

[append\(\)](#), [moved\(\)](#)

12.59.4.4 void QwtPicker::appended (const QPoint & *pos*) [signal, inherited]

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

Parameters

<i>pos</i>	Position of the appended point.
------------	---------------------------------

See also

[append\(\)](#), [moved\(\)](#)

12.59.4.5 `void QwtPicker::begin () [protected, virtual, inherited]`

Open a selection setting the state to active

See also

[isActive\(\)](#), [end\(\)](#), [append\(\)](#), [move\(\)](#)

Reimplemented in [QwtPlotZoomer](#).

12.59.4.6 `QwtPlotCanvas * QwtPlotPicker::canvas ()`

Return observed plot canvas.

12.59.4.7 `const QwtPlotCanvas * QwtPlotPicker::canvas () const`

Return Observed plot canvas.

12.59.4.8 `void QwtPicker::changed (const QwtPolygon & pa) [signal, inherited]`

A signal emitted when the active selection has been changed. This might happen when the observed widget is resized.

Parameters

<i>pa</i>	Changed selection
-----------	-------------------

See also

[stretchSelection\(\)](#)

12.59.4.9 `void QwtPicker::drawRubberBand (QPainter * painter) const [virtual, inherited]`

Draw a rubberband , depending on [rubberBand\(\)](#) and [selectionFlags\(\)](#)

Parameters

<i>painter</i>	Painter, initialized with clip rect
----------------	-------------------------------------

See also

[rubberBand\(\)](#), [RubberBand](#), [selectionFlags\(\)](#)

12.59.4.10 `void QwtPicker::drawTracker (QPainter * painter) const`
[virtual, inherited]

Draw the tracker

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[trackerRect\(\)](#), [trackerText\(\)](#)

12.59.4.11 `bool QwtPlotPicker::end (bool ok = true)` **[protected, virtual]**

Close a selection setting the state to inactive.

Parameters

<i>ok</i>	If true, complete the selection and emit selected signals otherwise discard the selection.
-----------	--

Returns

true if the selection is accepted, false otherwise

Reimplemented from [QwtPicker](#).

Reimplemented in [QwtPlotZoomer](#).

12.59.4.12 `bool QwtPicker::eventFilter (QObject * o, QEvent * e)`
[virtual, inherited]

Event filter.

When [isEnabled\(\)](#) == true all events of the observed widget are filtered. Mouse and keyboard events are translated into widgetMouse- and widgetKey- and widgetWheel-events. Paint and Resize events are handled to keep rubberband and tracker up to date.

See also

[event\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.59.4.13 `void QwtEventPattern::initKeyPattern ()` **[inherited]**

Set default mouse patterns.

See also

[KeyPatternCode](#)

12.59.4.14 `void QwtEventPattern::initMousePattern (int numButtons)`
[**inherited**]

Set default mouse patterns, depending on the number of mouse buttons

Parameters

<i>numButtons</i>	Number of mouse buttons (≤ 3)
-------------------	--------------------------------------

See also

[MousePatternCode](#)

12.59.4.15 `QwtDoublePoint QwtPlotPicker::invTransform (const QPoint & pos) const` [**protected**]

Translate a point from pixel into plot coordinates

Returns

Point in plot coordinates

See also

[QwtPlotPicker::transform\(\)](#)

12.59.4.16 `QwtDoubleRect QwtPlotPicker::invTransform (const QRect & rect) const` [**protected**]

Translate a rectangle from pixel into plot coordinates

Returns

Rectangle in plot coordinates

See also

[QwtPlotPicker::transform\(\)](#)

12.59.4.17 `bool QwtPicker::isActive () const` [**inherited**]

A picker is active between [begin\(\)](#) and [end\(\)](#).

Returns

true if the selection is active.

12.59.4.18 `bool QwtPicker::isEnabled () const [inherited]`**Returns**

true when enabled, false otherwise

See also

[setEnabled\(\)](#), [eventFilter\(\)](#)

12.59.4.19 `bool QwtEventPattern::keyMatch (uint pattern, const QKeyEvent * e) const [inherited]`

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.59.4.20 `bool QwtEventPattern::keyMatch (const KeyPattern & pattern, const QKeyEvent * e) const [protected, virtual, inherited]`

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Key event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.59.4.21 `const QwtArray< QwtEventPattern::KeyPattern > &
QwtEventPattern::keyPattern () const [inherited]`

Return key patterns.

12.59.4.22 `QwtArray< QwtEventPattern::KeyPattern > &
QwtEventPattern::keyPattern () [inherited]`

Return Key patterns.

12.59.4.23 `bool QwtEventPattern::mouseMatch (const MousePattern
& pattern, const QMouseEvent * e) const [protected,
virtual, inherited]`

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Mouse event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.59.4.24 `bool QwtEventPattern::mouseMatch (uint pattern, const
QMouseEvent * e) const [inherited]`

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.59.4.25 `const QwtArray< QwtEventPattern::MousePattern > &
QwtEventPattern::mousePattern () const [inherited]`

Return mouse patterns.

12.59.4.26 `QwtArray< QwtEventPattern::MousePattern > &
QwtEventPattern::mousePattern () [inherited]`

Return ,ouse patterns.

12.59.4.27 `void QwtPlotPicker::move (const QPoint & pos) [protected,
virtual]`

Move the last point of the selection

Parameters

<i>pos</i>	New position
------------	--------------

See also

[isActive](#), [begin\(\)](#), [end\(\)](#), [append\(\)](#)

Note

The [moved\(const QPoint &\)](#), [moved\(const QDoublePoint &\)](#) signals are emitted.

Reimplemented from [QwtPicker](#).

12.59.4.28 `void QwtPlotPicker::moved (const QwtDoublePoint & pos)`
[signal]

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

Parameters

<i>pos</i>	Position of the moved last point of the selection.
------------	--

See also

[move\(\)](#), [appended\(\)](#)

12.59.4.29 `void QwtPicker::moved (const QPoint & pos)` **[signal, inherited]**

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

Parameters

<i>pos</i>	Position of the moved last point of the selection.
------------	--

See also

[move\(\)](#), [appended\(\)](#)

12.59.4.30 `const QWidget * QwtPicker::parentWidget () const`
[inherited]

Return the parent widget, where the selection happens.

12.59.4.31 `QWidget * QwtPicker::parentWidget ()` **[inherited]**

Return the parent widget, where the selection happens.

12.59.4.32 `QRect QwtPicker::pickRect () const` **[virtual, inherited]**

Find the area of the observed widget, where selection might happen.

Returns

`QFrame::contentsRect()` if it is a `QFrame`, `QWidget::rect()` otherwise.

12.59.4.33 `const QwtPlot * QwtPlotPicker::plot () const`

Return plot widget, containing the observed plot canvas.

12.59.4.34 `QwtPlot * QwtPlotPicker::plot ()`

Return plot widget, containing the observed plot canvas.

12.59.4.35 `void QwtPicker::reset () [protected, virtual, inherited]`

Reset the state machine and terminate (end(false)) the selection

12.59.4.36 `QwtPicker::ResizeMode QwtPicker::resizeMode () const [inherited]`**Returns**

Resize mode

See also

[setResizeMode\(\)](#), [ResizeMode](#)

12.59.4.37 `QwtPicker::RubberBand QwtPicker::rubberBand () const [inherited]`**Returns**

Rubberband style

See also

[setRubberBand\(\)](#), [RubberBand](#), [rubberBandPen\(\)](#)

12.59.4.38 `QPen QwtPicker::rubberBandPen () const [inherited]`**Returns**

Rubberband pen

See also

[setRubberBandPen\(\)](#), [rubberBand\(\)](#)

12.59.4.39 `const QWidget * QwtPicker::rubberBandWidget () const`
[protected, inherited]

Returns

Widget displaying the rubberband

12.59.4.40 `QwtDoubleRect QwtPlotPicker::scaleRect () const`
[protected]

Return normalized bounding rect of the axes

See also

[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#).

12.59.4.41 `void QwtPlotPicker::selected (const QwtDoublePoint & pos)`
[signal]

A signal emitted in case of [selectionFlags\(\)](#) & PointSelection.

Parameters

<i>pos</i>	Selected point
------------	----------------

12.59.4.42 `void QwtPlotPicker::selected (const QwtDoubleRect & rect)`
[signal]

A signal emitted in case of [selectionFlags\(\)](#) & RectSelection.

Parameters

<i>rect</i>	Selected rectangle
-------------	--------------------

12.59.4.43 `void QwtPlotPicker::selected (const QwtArray< QwtDoublePoint > & pa) [signal]`

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

Parameters

<i>pa</i>	Selected points
-----------	-----------------

12.59.4.44 `void QwtPicker::selected (const QwtPolygon & pa) [signal, inherited]`

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

Parameters

<i>pa</i>	Selected points
-----------	-----------------

12.59.4.45 `const QwtPolygon & QwtPicker::selection () const [inherited]`

Return Selected points.

12.59.4.46 `int QwtPicker::selectionFlags () const [inherited]`

Returns

Selection flags, an Or'd value of SelectionType, RectSelectionType and SelectionMode.

See also

[setSelectionFlags\(\)](#), [SelectionType](#), [RectSelectionType](#), [SelectionMode](#)

12.59.4.47 `void QwtPlotPicker::setAxis (int xAxis, int yAxis) [virtual]`

Set the x and y axes of the picker

Parameters

<i>xAxis</i>	X axis
<i>yAxis</i>	Y axis

Reimplemented in [QwtPlotZoomer](#).

12.59.4.48 void QwtPicker::setEnabled (bool *enabled*) [virtual, inherited]

En/disable the picker.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>enabled</i>	true or false
----------------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.59.4.49 void QwtEventPattern::setKeyPattern (uint *pattern*, int *key*, int *state* = Qt::NoButton) [inherited]

Change one key pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>key</i>	Key
<i>state</i>	State

See also

[QKeyEvent](#)

12.59.4.50 void QwtEventPattern::setKeyPattern (const QwtArray< KeyPattern > & *pattern*) [inherited]

Change the key event patterns.

12.59.4.51 void QwtEventPattern::setMousePattern (const QwtArray< MousePattern > & *pattern*) [inherited]

Change the mouse event patterns.

12.59.4.52 void QwtEventPattern::setMousePattern (uint *pattern*, int *button*, int *state* = Qt::NoButton) [inherited]

Change one mouse pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>button</i>	Button
<i>state</i>	State

See also

QMouseEvent

12.59.4.53 void QwtPicker::setResizeMode (ResizeMode *mode*)
[virtual, inherited]

Set the resize mode.

The resize mode controls what to do with the selected points of an active selection when the observed widget is resized.

Stretch means the points are scaled according to the new size, KeepSize means the points remain unchanged.

The default mode is Stretch.

Parameters

<i>mode</i>	Resize mode
-------------	-------------

See also

[resizeMode\(\)](#), [ResizeMode](#)

12.59.4.54 void QwtPicker::setRubberBand (RubberBand *rubberBand*)
[virtual, inherited]

Set the rubberband style

Parameters

<i>rubberBand</i>	Rubberband style The default value is NoRubberBand.
-------------------	---

See also

[rubberBand\(\)](#), [RubberBand](#), [setRubberBandPen\(\)](#)

12.59.4.55 void QwtPicker::setRubberBandPen (const QPen & *pen*)
[virtual, inherited]

Set the pen for the rubberband

Parameters

<i>pen</i>	Rubberband pen
------------	----------------

See also

[rubberBandPen\(\)](#), [setRubberBand\(\)](#)

12.59.4.56 void QwtPicker::setSelectionFlags (int *flags*) [virtual, inherited]

Set the selection flags

Parameters

<i>flags</i>	Or'd value of SelectionType, RectSelectionType and SelectionMode. The default value is NoSelection.
--------------	---

See also

[selectionFlags\(\)](#), [SelectionType](#), [RectSelectionType](#), [SelectionMode](#)

Reimplemented in [QwtPlotZoomer](#).

12.59.4.57 void QwtPicker::setTrackerFont (const QFont & *font*)
[virtual, inherited]

Set the font for the tracker

Parameters

<i>font</i>	Tracker font
-------------	--------------

See also

[trackerFont\(\)](#), [setTrackerMode\(\)](#), [setTrackerPen\(\)](#)

12.59.4.58 void QwtPicker::setTrackerMode (DisplayMode *mode*)
[virtual, inherited]

Set the display mode of the tracker.

A tracker displays information about current position of the cursor as a string. The

display mode controls if the tracker has to be displayed whenever the observed widget has focus and cursor (AlwaysOn), never (AlwaysOff), or only when the selection is active (ActiveOnly).

Parameters

<i>mode</i>	Tracker display mode
-------------	----------------------

Warning

In case of AlwaysOn, mouseTracking will be enabled for the observed widget.

See also

[trackerMode\(\)](#), [DisplayMode](#)

12.59.4.59 `void QwtPicker::setTrackerPen (const QPen & pen)` [**virtual**, **inherited**]

Set the pen for the tracker

Parameters

<i>pen</i>	Tracker pen
------------	-------------

See also

[trackerPen\(\)](#), [setTrackerMode\(\)](#), [setTrackerFont\(\)](#)

12.59.4.60 `QwtPickerMachine * QwtPicker::stateMachine (int flags) const` [**protected**, **virtual**, **inherited**]

Create a state machine depending on the selection flags.

- PointSelection | ClickSelection
QwtPickerClickPointMachine()
- PointSelection | DragSelection
QwtPickerDragPointMachine()
- RectSelection | ClickSelection
QwtPickerClickRectMachine()
- RectSelection | DragSelection
QwtPickerDragRectMachine()
- PolygonSelection
QwtPickerPolygonMachine()

See also

[setSelectionFlags\(\)](#)

12.59.4.61 `void QwtPicker::stretchSelection (const QSize & oldSize, const QSize & newSize)` [**protected**, **virtual**, **inherited**]

Scale the selection by the ratios of *oldSize* and *newSize* The [changed\(\)](#) signal is emitted.

Parameters

<i>oldSize</i>	Previous size
<i>newSize</i>	Current size

See also

[ResizeMode](#), [setResizeMode\(\)](#), [resizeMode\(\)](#)

12.59.4.62 `QFont QwtPicker::trackerFont () const` [**inherited**]

Returns

Tracker font

See also

[setTrackerFont\(\)](#), [trackerMode\(\)](#), [trackerPen\(\)](#)

12.59.4.63 `QwtPicker::DisplayMode QwtPicker::trackerMode () const` [**inherited**]

Returns

Tracker display mode

See also

[setTrackerMode\(\)](#), [DisplayMode](#)

12.59.4.64 `QPen QwtPicker::trackerPen () const` [**inherited**]

Returns

Tracker pen

See also

[setTrackerPen\(\)](#), [trackerMode\(\)](#), [trackerFont\(\)](#)

12.59.4.65 QPoint QwtPicker::trackerPosition () const [inherited]**Returns**

Current position of the tracker

12.59.4.66 QRect QwtPicker::trackerRect (const QFont & *font*) const [inherited]

Calculate the bounding rectangle for the tracker text from the current position of the tracker

Parameters

<i>font</i>	Font of the tracker text
-------------	--------------------------

Returns

Bounding rectangle of the tracker text

See also

[trackerPosition\(\)](#)

12.59.4.67 QText QwtPlotPicker::trackerText (const QwtDoublePoint & *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

<i>pos</i>	Position
------------	----------

Returns

Position string

12.59.4.68 `QwtText QwtPlotPicker::trackerText (const QPoint & pos) const`
[protected, virtual]

Translate a pixel position into a position string

Parameters

<i>pos</i> Position in pixel coordinates
--

Returns

Position string

Reimplemented from [QwtPicker](#).

12.59.4.69 `const QWidget * QwtPicker::trackerWidget () const`
[protected, inherited]

Returns

Widget displaying the tracker text

12.59.4.70 `QPoint QwtPlotPicker::transform (const QwtDoublePoint & pos)`
`const` [protected]

Translate a point from plot into pixel coordinates

Returns

Point in pixel coordinates

See also

[QwtPlotPicker::invTransform\(\)](#)

12.59.4.71 `QRect QwtPlotPicker::transform (const QwtDoubleRect & rect)`
`const` [protected]

Translate a rectangle from plot into pixel coordinates

Returns

Rectangle in pixel coordinates

See also

[QwtPlotPicker::invTransform\(\)](#)

12.59.4.72 `void QwtPicker::transition (const QEvent * e) [protected, virtual, inherited]`

Passes an event to the state machine and executes the resulting commands. Append and Move commands use the current position of the cursor (`QCursor::pos()`).

Parameters

<i>e</i>	Event
----------	-------

12.59.4.73 `void QwtPicker::updateDisplay () [protected, virtual, inherited]`

Update the state of rubberband and tracker label.

12.59.4.74 `void QwtPicker::widgetKeyPressEvent (QKeyEvent * ke) [protected, virtual, inherited]`

Handle a key press event for the observed widget.

Selections can be completely done by the keyboard. The arrow keys move the cursor, the abort key aborts a selection. All other keys are handled by the current state machine.

See also

[QwtPicker](#), [selectionFlags\(\)](#), [eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#), [stateMachine\(\)](#), [QwtEventPattern::KeyPatternCode](#)

Reimplemented in [QwtPlotZoomer](#).

12.59.4.75 `void QwtPicker::widgetKeyReleaseEvent (QKeyEvent * ke) [protected, virtual, inherited]`

Handle a key release event for the observed widget.

Passes the event to the state machine.

See also

[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [stateMachine\(\)](#)

12.59.4.76 `void QwtPicker::widgetLeaveEvent (QEvent *)` [**protected**, **virtual**, **inherited**]

Handle a leave event for the observed widget.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.59.4.77 `void QwtPicker::widgetMouseDoubleClickEvent (QMouseEvent * me)` [**protected**, **virtual**, **inherited**]

Handle mouse double click event for the observed widget.

Empty implementation, does nothing.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.59.4.78 `void QwtPicker::widgetMouseMoveEvent (QMouseEvent * e)` [**protected**, **virtual**, **inherited**]

Handle a mouse move event for the observed widget.

Move the last point of the selection in case of [isActive\(\)](#) == true

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.59.4.79 `void QwtPicker::widgetMousePressEvent (QMouseEvent * e)` [**protected**, **virtual**, **inherited**]

Handle a mouse press event for the observed widget.

Begin and/or end a selection depending on the selection flags.

See also

[QwtPicker](#), [selectionFlags\(\)](#), [eventFilter\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.59.4.80 void QwtPicker::widgetMouseReleaseEvent (QMouseEvent * *e*)
[protected, virtual, inherited]

Handle a mouse release event for the observed widget.

End a selection depending on the selection flags.

See also

[QwtPicker](#), [selectionFlags\(\)](#)
[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

Reimplemented in [QwtPlotZoomer](#).

12.59.4.81 void QwtPicker::widgetWheelEvent (QWheelEvent * *e*)
[protected, virtual, inherited]

Handle a wheel event for the observed widget.

Move the last point of the selection in case of [isActive\(\)](#) == true

See also

[eventFilter\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.59.4.82 int QwtPlotPicker::xAxis () const

Return x axis.

12.59.4.83 int QwtPlotPicker::yAxis () const

Return y axis.

12.60 QwtPlotPrintFilter Class Reference

A base class for plot print filters.

```
#include <qwt_plot_printfilter.h>
```

Public Types

- enum [Item](#) {
 Title,
 Legend,
 Curve,
 CurveSymbol,
 Marker,
 MarkerSymbol,
 MajorGrid,
 MinorGrid,
 CanvasBackground,
 AxisScale,
 AxisTitle,
 WidgetBackground }
• enum [Options](#) {
 PrintMargin = 1,
 PrintTitle = 2,
 PrintLegend = 4,
 PrintGrid = 8,
 PrintBackground = 16,
 PrintFrameWithScales = 32,
 PrintAll = ~PrintFrameWithScales }

Public Member Functions

- virtual void [apply](#) ([QwtPlot](#) *) const
- virtual void [apply](#) ([QwtPlotItem](#) *) const
- virtual [QColor](#) [color](#) (const [QColor](#) &, [Item](#) item) const
- virtual [QFont](#) [font](#) (const [QFont](#) &, [Item](#) item) const
- int [options](#) () const
- [QwtPlotPrintFilter](#) ()
- virtual void [reset](#) ([QwtPlotItem](#) *) const
- virtual void [reset](#) ([QwtPlot](#) *) const
- void [setOptions](#) (int options)
- virtual [~QwtPlotPrintFilter](#) ()

12.60.1 Detailed Description

A base class for plot print filters. A print filter can be used to customize [QwtPlot::print\(\)](#).

Deprecated

In Qwt 5.0 the design of [QwtPlot](#) allows/recommends writing individual Qwt-PlotItems, that are not known to [QwtPlotPrintFilter](#). So this concept is outdated and [QwtPlotPrintFilter](#) will be removed/replaced in Qwt 6.x.

12.60.2 Member Enumeration Documentation

12.60.2.1 enum QwtPlotPrintFilter::Item

Print items.

12.60.2.2 enum QwtPlotPrintFilter::Options

Print options.

12.60.3 Constructor & Destructor Documentation

12.60.3.1 QwtPlotPrintFilter::QwtPlotPrintFilter () [explicit]

Sets filter options to PrintAll

12.60.3.2 QwtPlotPrintFilter::~QwtPlotPrintFilter () [virtual]

Destructor.

12.60.4 Member Function Documentation

12.60.4.1 void QwtPlotPrintFilter::apply (QwtPlot * *plot*) const [virtual]

Change color and fonts of a plot

See also

[apply\(\)](#)

12.60.4.2 QColor QwtPlotPrintFilter::color (const QColor & *c*, Item *item*) const [virtual]

Modifies a color for printing.

Parameters

<i>c</i>	Color to be modified
<i>item</i>	Type of item where the color belongs

Returns

Modified color.

In case of `!(QwtPlotPrintFilter::options() & PrintBackground)` MajorGrid is modified to `Qt::darkGray`, MinorGrid to `Qt::gray`. All other colors are returned unmodified.

12.60.4.3 QFont QwtPlotPrintFilter::font (const QFont & *f*, Item *item*) const [virtual]

Modifies a font for printing.

Parameters

<i>f</i>	Font to be modified
<i>item</i>	Type of item where the font belongs

All fonts are returned unmodified

12.60.4.4 int QwtPlotPrintFilter::options () const

Get plot print options.

See also

[setOptions\(\)](#)

12.60.4.5 void QwtPlotPrintFilter::reset (QwtPlot * *plot*) const [virtual]

Reset color and fonts of a plot

See also

[apply\(\)](#)

12.60.4.6 void QwtPlotPrintFilter::setOptions (int *options*)

Set plot print options.

Parameters

<i>options</i>	Or'd QwtPlotPrintFilter::Options values
----------------	---

See also

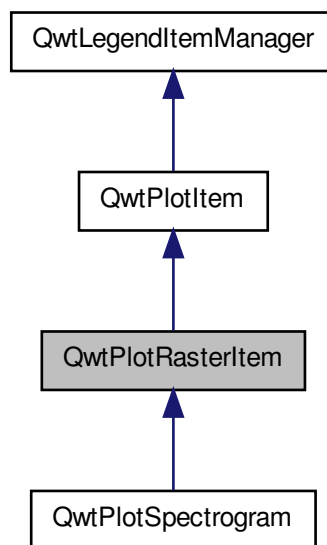
[options\(\)](#)

12.61 QwtPlotRasterItem Class Reference

A class, which displays raster data.

```
#include <qwt_plot_rasteritem.h>
```

Inheritance diagram for QwtPlotRasterItem:



Public Types

- enum [CachePolicy](#) {

```

    NoCache,
    PaintCache,
    ScreenCache }
• enum ItemAttribute {
    Legend = 1,
    AutoScale = 2 }
• enum RenderHint { RenderAntialiased = 1 }
• enum RttiValues {
    Rtti_PlotItem = 0,
    Rtti_PlotGrid,
    Rtti_PlotScale,
    Rtti_PlotMarker,
    Rtti_PlotCurve,
    Rtti_PlotHistogram,
    Rtti_PlotSpectrogram,
    Rtti_PlotSVG,
    Rtti_PlotUserItem = 1000 }

```

Public Member Functions

- int [alpha](#) () const
- void [attach](#) ([QwtPlot](#) *plot)
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- [CachePolicy](#) [cachePolicy](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &rect) const
- void [hide](#) ()
- void [invalidateCache](#) ()
- [QwtDoubleRect](#) [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QRect](#) &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual [QWidget](#) * [legendItem](#) () const
- [QRect](#) [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- [QwtPlot](#) * [plot](#) () const
- [QwtPlotRasterItem](#) (const [QString](#) &title=[QString::null](#))
- [QwtPlotRasterItem](#) (const [QwtText](#) &title)
- virtual [QSize](#) [rasterHint](#) (const [QwtDoubleRect](#) &) const
- virtual int [rtti](#) () const
- [QwtDoubleRect](#) [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAlpha](#) (int alpha)
- void [setAxis](#) (int xAxis, int yAxis)

- void [setCachePolicy](#) ([CachePolicy](#))
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setTitle](#) (const QString &title)
- void [setTitle](#) (const [QwtText](#) &title)
- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setYAxis](#) (int axis)
- void [setZ](#) (double z)
- void [show](#) ()
- bool [testItemAttribute](#) ([ItemAttribute](#)) const
- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- QRect [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const QwtDoubleRect &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- int [xAxis](#) () const
- int [yAxis](#) () const
- double [z](#) () const
- virtual [~QwtPlotRasterItem](#) ()

Protected Member Functions

- virtual QImage [renderImage](#) (const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const QwtDoubleRect &area) const =0

12.61.1 Detailed Description

A class, which displays raster data. Raster data is a grid of pixel values, that can be represented as a QImage. It is used for many types of information like spectrograms, cartograms, geographical maps ...

Often a plot has several types of raster data organized in layers. (f.e a geographical map, with weather statistics). Using [setAlpha\(\)](#) raster items can be stacked easily.

[QwtPlotRasterItem](#) is only implemented for images of the following formats: QImage::Format_Indexed8, QImage::Format_ARGB32.

See also

[QwtPlotSpectrogram](#)

12.61.2 Member Enumeration Documentation

12.61.2.1 enum QwtPlotRasterItem::CachePolicy

- NoCache
[renderImage\(\)](#) is called, whenever the item has to be repainted
- PaintCache
[renderImage\(\)](#) is called, whenever the image cache is not valid, or the scales, or the size of the canvas has changed. This type of cache is only useful for improving the performance of hide/show operations. All other situations are already handled by the plot canvas cache.
- ScreenCache
The screen cache is an image in size of the screen. As long as the scales don't change the target image is scaled from the cache. This might improve the performance when resizing the plot widget, but suffers from scaling effects.

The default policy is NoCache

12.61.2.2 enum QwtPlotItem::ItemAttribute [inherited]

Plot Item Attributes

- Legend
The item is represented on the legend.
- AutoScale
The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.61.2.3 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.61.2.4 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.61.3 Constructor & Destructor Documentation

12.61.3.1 `QwtPlotRasterItem::QwtPlotRasterItem (const QString & title = QString::null) [explicit]`

Constructor.

12.61.3.2 `QwtPlotRasterItem::QwtPlotRasterItem (const QwtText & title) [explicit]`

Constructor.

12.61.3.3 `QwtPlotRasterItem::~~QwtPlotRasterItem () [virtual]`

Destructor.

12.61.4 Member Function Documentation

12.61.4.1 `int QwtPlotRasterItem::alpha () const`

Returns

Alpha value of the raster item

See also

[setAlpha\(\)](#)

12.61.4.2 `void QwtPlotItem::attach (QwtPlot * plot) [inherited]`

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.61.4.3 `QwtDoubleRect QwtPlotItem::boundingRect () const`
[virtual, inherited]

Returns

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

Reimplemented in [QwtPlotCurve](#), [QwtPlotMarker](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.61.4.4 `QwtPlotRasterItem::CachePolicy QwtPlotRasterItem::cachePolicy () const`

Returns

Cache policy

See also

[CachePolicy](#), [setCachePolicy\(\)](#)

12.61.4.5 `void QwtPlotItem::detach () [inline, inherited]`

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with. [detach\(\)](#) is equivalent to calling `attach(NULL)`

See also

[attach\(QwtPlot* plot \)](#)

12.61.4.6 `void QwtPlotRasterItem::draw (QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & canvasRect) const` [virtual]

Draw the raster data.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	X-Scale Map
<i>yMap</i>	Y-Scale Map
<i>canvasRect</i>	Contents rect of the plot canvas

Implements [QwtPlotItem](#).

Reimplemented in [QwtPlotSpectrogram](#).

12.61.4.7 void QwtPlotItem::hide () **[inherited]**

Hide the item.

12.61.4.8 void QwtPlotRasterItem::invalidateCache ()

Invalidate the paint cache

See also

[setCachePolicy\(\)](#)

12.61.4.9 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *rect*) const **[inherited]**

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.61.4.10 bool QwtPlotItem::isVisible () const **[inherited]**

Returns

true if visible

See also

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

12.61.4.11 void QwtPlotItem::itemChanged () [virtual, inherited]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.61.4.12 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.61.4.13 QRect QwtPlotItem::paintRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.61.4.14 QwtPlot * QwtPlotItem::plot () const [inherited]

Return attached plot.

12.61.4.15 QSize QwtPlotRasterItem::rasterHint (const QwtDoubleRect &) const [virtual]

Returns the recommended raster for a given rect.

E.g. the raster hint can be used to limit the resolution of the image that is rendered.

The default implementation returns an invalid size (QSize()), what means: no hint.

Reimplemented in [QwtPlotSpectrogram](#).

12.61.4.16 virtual QImage QwtPlotRasterItem::renderImage (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QwtDoubleRect & area) const [protected, pure virtual]

Renders an image for an area

The format of the image must be QImage::Format_Indexed8, QImage::Format_RGB32 or QImage::Format_ARGB32

Parameters

<i>xMap</i>	Maps x-values into pixel coordinates.
<i>yMap</i>	Maps y-values into pixel coordinates.
<i>area</i>	Requested area for the image in scale coordinates

Implemented in [QwtPlotSpectrogram](#).

12.61.4.17 int QwtPlotItem::rtti () const [virtual, inherited]

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

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 [QwtPlotCurve](#), [QwtPlotGrid](#), [QwtPlotMarker](#), [QwtPlotScaleItem](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.61.4.18 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const [inherited]

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.61.4.19 void QwtPlotRasterItem::setAlpha (int *alpha*)

Set an alpha value for the raster data.

Often a plot has several types of raster data organized in layers. (f.e a geographical map, with weather statistics). Using [setAlpha\(\)](#) raster items can be stacked easily.

The alpha value is a value [0, 255] to control the transparency of the image. 0 represents a fully transparent color, while 255 represents a fully opaque color.

Parameters

<i>alpha</i>	Alpha value
--------------	-------------

- $\alpha \geq 0$
All alpha values of the pixels returned by [renderImage\(\)](#) will be set to alpha, beside those with an alpha value of 0 (invalid pixels).
- $\alpha < 0$ The alpha values returned by [renderImage\(\)](#) are not changed.

The default alpha value is -1.

See also

[alpha\(\)](#)

12.61.4.20 void QwtPlotItem::setAxis (int *xAxis*, int *yAxis*) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.61.4.21 void QwtPlotRasterItem::setCachePolicy (QwtPlotRasterItem::CachePolicy *policy*)

Change the cache policy

The default policy is NoCache

Parameters

<i>policy</i>	Cache policy
---------------	--------------

See also

[CachePolicy](#), [cachePolicy\(\)](#)

12.61.4.22 void QwtPlotItem::setItemAttribute (ItemAttribute *attribute*, bool *on = true*) [inherited]

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.61.4.23 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on* = *true*) [inherited]

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.61.4.24 void QwtPlotItem::setTitle (const QString & *title*) [inherited]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.61.4.25 void QwtPlotItem::setTitle (const QwtText & *title*) [inherited]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.61.4.26 void QwtPlotItem::setVisible (bool *on*) [virtual, inherited]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also

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

12.61.4.27 void QwtPlotItem::setXAxis (int *axis*) [inherited]

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also

[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)

12.61.4.28 void QwtPlotItem::setYAxis (int *axis*) [inherited]

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also

[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)

12.61.4.29 void QwtPlotItem::setZ (double *z*) [inherited]

Set the *z* value.

Plot items are painted in increasing *z*-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also

[z\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.61.4.30 void QwtPlotItem::show () [inherited]

Show the item.

12.61.4.31 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*) const [inherited]

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also

[setItemAttribute\(\)](#), [ItemAttribute](#)

12.61.4.32 bool QwtPlotItem::testRenderHint (RenderHint *hint*) const [inherited]

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.61.4.33 const QwtText & QwtPlotItem::title () const [inherited]**Returns**

Title of the item

See also

[setTitle\(\)](#)

12.61.4.34 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap,
const QwtScaleMap & yMap, const QwtDoubleRect & rect) const`
[**inherited**]

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.61.4.35 `void QwtPlotItem::updateLegend (QwtLegend * legend) const`
[**virtual, inherited**]

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also

[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

12.61.4.36 `void QwtPlotItem::updateScaleDiv (const QwtScaleDiv & , const
QwtScaleDiv &)` [virtual, inherited]

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 [QwtPlotGrid\(\)](#)) have to reimplement [updateScaleDiv\(\)](#)

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented in [QwtPlotGrid](#), and [QwtPlotScaleItem](#).

12.61.4.37 int QwtPlotItem::xAxis () const [inherited]

Return xAxis.

12.61.4.38 int QwtPlotItem::yAxis () const [inherited]

Return yAxis.

12.61.4.39 double QwtPlotItem::z () const [inherited]

Plot items are painted in increasing z-order.

Returns

[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.62 QwtPlotRescaler Class Reference

[QwtPlotRescaler](#) takes care of fixed aspect ratios for plot scales.

```
#include <qwt_plot_rescaler.h>
```

Public Types

- enum **ExpandingDirection** {
 ExpandUp,
 ExpandDown,
 ExpandBoth }

- enum [RescalePolicy](#) {
Fixed,
Expanding,
Fitting }

Public Member Functions

- double [aspectRatio](#) (int axis) const
- [QwtPlotCanvas](#) * [canvas](#) ()
- const [QwtPlotCanvas](#) * [canvas](#) () const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- ExpandingDirection [expandingDirection](#) (int axis) const
- [QwtDoubleInterval](#) [intervalHint](#) (int axis) const
- bool [isEnabled](#) () const
- [QwtPlot](#) * [plot](#) ()
- const [QwtPlot](#) * [plot](#) () const
- [QwtPlotRescaler](#) ([QwtPlotCanvas](#) *, int referenceAxis=[QwtPlot::xBottom](#), [RescalePolicy](#)=[Expanding](#))
- int [referenceAxis](#) () const
- void [rescale](#) () const
- [RescalePolicy](#) [rescalePolicy](#) () const
- void [setAspectRatio](#) (int axis, double ratio)
- void [setAspectRatio](#) (double ratio)
- void [setEnabled](#) (bool)
- void [setExpandingDirection](#) (int axis, ExpandingDirection)
- void [setExpandingDirection](#) (ExpandingDirection)
- void [setIntervalHint](#) (int axis, const [QwtDoubleInterval](#) &)
- void [setReferenceAxis](#) (int axis)
- void [setRescalePolicy](#) ([RescalePolicy](#))
- virtual [~QwtPlotRescaler](#) ()

Protected Member Functions

- virtual void [canvasResizeEvent](#) (QResizeEvent *)
- [QwtDoubleInterval](#) [expandInterval](#) (const [QwtDoubleInterval](#) &, double width, ExpandingDirection) const
- virtual [QwtDoubleInterval](#) [expandScale](#) (int axis, const QSize &oldSize, const QSize &newSize) const
- [QwtDoubleInterval](#) [interval](#) (int axis) const
- Qt::Orientation [orientation](#) (int axis) const
- virtual void [rescale](#) (const QSize &oldSize, const QSize &newSize) const
- virtual [QwtDoubleInterval](#) [syncScale](#) (int axis, const [QwtDoubleInterval](#) &reference, const QSize &size) const
- virtual void [updateScales](#) ([QwtDoubleInterval](#) intervals[[QwtPlot::axisCnt](#)]) const

12.62.1 Detailed Description

[QwtPlotRescaler](#) takes care of fixed aspect ratios for plot scales. [QwtPlotRescaler](#) autoadjusts the axes of a [QwtPlot](#) according to fixed aspect ratios.

12.62.2 Member Enumeration Documentation

12.62.2.1 enum QwtPlotRescaler::RescalePolicy

Rescale Policy.

The rescale policy defines how to rescale the reference axis and their depending axes.

- Fixed

The interval of the reference axis remains unchanged, when the geometry of the canvas changes. All other axes will be adjusted according to their aspect ratio.

- Expanding

The interval of the reference axis will be shrunk/expanded, when the geometry of the canvas changes. All other axes will be adjusted according to their aspect ratio.

The interval, that is represented by one pixel is fixed.

- Fitting

The intervals of the axes are calculated, so that all axes include their minimal interval.

12.62.3 Constructor & Destructor Documentation

12.62.3.1 QwtPlotRescaler::QwtPlotRescaler (QwtPlotCanvas * *canvas*, int *referenceAxis* = *QwtPlot::xBottom*, RescalePolicy *policy* = *Expanding*) [explicit]

Constructor

Parameters

<i>canvas</i>	Canvas
<i>referenceAxis</i>	Reference axis, see RescalePolicy
<i>policy</i>	Rescale policy

See also

[setRescalePolicy\(\)](#), [setReferenceAxis\(\)](#)

12.62.3.2 QwtPlotRescaler::~~QwtPlotRescaler () [virtual]

Destructor.

12.62.4 Member Function Documentation**12.62.4.1 double QwtPlotRescaler::aspectRatio (int *axis*) const**

Return aspect ratio between an axis and the reference axis.

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
-------------	------------------------------------

See also

[setAspectRatio\(\)](#)

12.62.4.2 const QwtPlotCanvas * QwtPlotRescaler::canvas () const**Returns**

plot canvas

12.62.4.3 QwtPlotCanvas * QwtPlotRescaler::canvas ()**Returns**

plot canvas

12.62.4.4 bool QwtPlotRescaler::eventFilter (QObject * *o*, QEvent * *e*) [virtual]

Event filter for the plot canvas.

**12.62.4.5 QwtPlotRescaler::ExpandingDirection
QwtPlotRescaler::expandingDirection (int *axis*) const**

Return direction in which an axis should be expanded

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
-------------	------------------------------------

See also

[setExpandingDirection\(\)](#)

12.62.4.6 `QwtDoubleInterval QwtPlotRescaler::expandInterval (const QwtDoubleInterval & interval, double width, ExpandingDirection direction) const` **[protected]**

Expand the interval

Parameters

<i>interval</i>	Interval to be expanded
<i>width</i>	Distance to be added to the interval
<i>direction</i>	Direction of the expand operation

Returns

Expanded interval

12.62.4.7 `QwtDoubleInterval QwtPlotRescaler::expandScale (int axis, const QSize & oldSize, const QSize & newSize) const` **[protected, virtual]**

Calculate the new scale interval of a plot axis

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
<i>oldSize</i>	Previous size of the canvas
<i>newSize</i>	New size of the canvas

Returns

Calculated new interval for the axis

12.62.4.8 `QwtDoubleInterval QwtPlotRescaler::interval (int axis) const` **[protected]**

Return interval of an axis

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
-------------	------------------------------------

12.62.4.9 `bool QwtPlotRescaler::isEnabled () const`**Returns**

true when enabled, false otherwise

See also

[setEnabled](#), [eventFilter\(\)](#)

12.62.4.10 `Qt::Orientation QwtPlotRescaler::orientation (int axis) const`
[protected]

Return orientation of an axis

Parameters

<i>axis</i> Axis index (see <code>QwtPlot::AxisId</code>)

12.62.4.11 `QwtPlot * QwtPlotRescaler::plot ()`**Returns**

plot widget

12.62.4.12 `const QwtPlot * QwtPlotRescaler::plot () const`**Returns**

plot widget

12.62.4.13 `int QwtPlotRescaler::referenceAxis () const`**Returns**

Reference axis (see `RescalePolicy`)

See also

[setReferenceAxis\(\)](#)

12.62.4.14 void QwtPlotRescaler::rescale () const

Adjust the plot axes scales.

12.62.4.15 void QwtPlotRescaler::rescale (const QSize & *oldSize*, const QSize & *newSize*) const [protected, virtual]

Adjust the plot axes scales

Parameters

<i>oldSize</i>	Previous size of the canvas
<i>newSize</i>	New size of the canvas

12.62.4.16 QwtPlotRescaler::RescalePolicy QwtPlotRescaler::rescalePolicy () const**Returns**

Rescale policy

See also

[setRescalePolicy\(\)](#)

12.62.4.17 void QwtPlotRescaler::setAspectRatio (int *axis*, double *ratio*)

Set the aspect ratio between the scale of the reference axis and another scale. The default ratio is 1.0

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
<i>ratio</i>	Aspect ratio

See also

[aspectRatio\(\)](#)

12.62.4.18 void QwtPlotRescaler::setAspectRatio (double *ratio*)

Set the aspect ratio between the scale of the reference axis and the other scales. The default ratio is 1.0

Parameters

<i>ratio</i>	Aspect ratio
--------------	--------------

See also

[aspectRatio\(\)](#)

12.62.4.19 void QwtPlotRescaler::setEnabled (bool *on*)

En/disable the rescaler.

When enabled is true an event filter is installed for the canvas, otherwise the event filter is removed.

Parameters

<i>on</i>	true or false
-----------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.62.4.20 void QwtPlotRescaler::setExpandingDirection (int *axis*, ExpandingDirection *direction*)

Set the direction in which an axis should be expanded

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
<i>direction</i>	Direction

See also

[expandingDirection\(\)](#)

12.62.4.21 void QwtPlotRescaler::setExpandingDirection (ExpandingDirection *direction*)

Set the direction in which all axis should be expanded

Parameters

<i>direction</i>	Direction
------------------	-----------

See also

[expandingDirection\(\)](#)

12.62.4.22 void QwtPlotRescaler::setReferenceAxis (int *axis*)

Set the reference axis (see RescalePolicy)

Parameters

<i>axis</i>	Axis index (QwtPlot::Axis)
-------------	--

See also

[referenceAxis\(\)](#)

12.62.4.23 void QwtPlotRescaler::setRescalePolicy (RescalePolicy *policy*)

Change the rescale policy

Parameters

<i>policy</i>	Rescale policy
---------------	----------------

See also

[rescalePolicy\(\)](#)

12.62.4.24 QwtDoubleInterval QwtPlotRescaler::syncScale (int *axis*, const QwtDoubleInterval & *reference*, const QSize & *size*) const [protected, virtual]

Synchronize an axis scale according to the scale of the reference axis

Parameters

<i>axis</i>	Axis index (see QwtPlot::AxisId)
<i>reference</i>	Interval of the reference axis
<i>size</i>	Size of the canvas

12.62.4.25 void QwtPlotRescaler::updateScales (QwtDoubleInterval *intervals*[*QwtPlot::axisCnt*]) const [protected, virtual]

Update the axes scales

Parameters

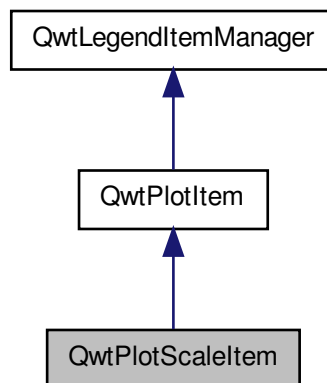
<i>intervals</i>	Scale intervals
------------------	-----------------

12.63 QwtPlotScaleItem Class Reference

A class which draws a scale inside the plot canvas.

```
#include <qwt_plot_scaleitem.h>
```

Inheritance diagram for QwtPlotScaleItem:

**Public Types**

- enum [ItemAttribute](#) {
Legend = 1,
AutoScale = 2 }
- enum [RenderHint](#) { **RenderAntialiased** = 1 }
- enum [RttiValues](#) {
Rtti_PlotItem = 0,
Rtti_PlotGrid,
Rtti_PlotScale,
Rtti_PlotMarker,
Rtti_PlotCurve,
Rtti_PlotHistogram,
Rtti_PlotSpectrogram,

```

Rtti_PlotSVG,
Rtti_PlotUserItem = 1000 }

```

Public Member Functions

- void [attach](#) ([QwtPlot](#) *plot)
- int [borderDistance](#) () const
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &rect) const
- [QFont](#) [font](#) () const
- void [hide](#) ()
- [QwtDoubleRect](#) [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QRect](#) &) const
- bool [isScaleDivFromAxis](#) () const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual [QWidget](#) * [legendItem](#) () const
- [QRect](#) [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- [QPalette](#) [palette](#) () const
- [QwtPlot](#) * [plot](#) () const
- double [position](#) () const
- [QwtPlotScaleItem](#) ([QwtScaleDraw::Alignment](#)=[QwtScaleDraw::BottomScale](#), const double pos=0.0)
- virtual int [rtti](#) () const
- const [QwtScaleDiv](#) & [scaleDiv](#) () const
- const [QwtScaleDraw](#) * [scaleDraw](#) () const
- [QwtScaleDraw](#) * [scaleDraw](#) ()
- [QwtDoubleRect](#) [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAlignment](#) ([QwtScaleDraw::Alignment](#))
- void [setAxis](#) (int xAxis, int yAxis)
- void [setBorderDistance](#) (int numPixels)
- void [setFont](#) (const [QFont](#) &)
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setPalette](#) (const [QPalette](#) &)
- void [setPosition](#) (double pos)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setScaleDiv](#) (const [QwtScaleDiv](#) &)
- void [setScaleDivFromAxis](#) (bool on)
- void [setScaleDraw](#) ([QwtScaleDraw](#) *)
- void [setTitle](#) (const [QString](#) &title)
- void [setTitle](#) (const [QwtText](#) &title)
- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setYAxis](#) (int axis)

- void [setZ](#) (double z)
- void [show](#) ()
- bool [testItemAttribute](#) ([ItemAttribute](#)) const
- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- QRect [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QwtDoubleRect](#) &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- int [xAxis](#) () const
- int [yAxis](#) () const
- double [z](#) () const
- virtual [~QwtPlotScaleItem](#) ()

12.63.1 Detailed Description

A class which draws a scale inside the plot canvas. [QwtPlotScaleItem](#) can be used to draw an axis inside the plot canvas. It might be synchronized to one of the axis of the plot, but can also display its own ticks and labels.

It is allowed to synchronize the scale item with a disabled axis. In plots with vertical and horizontal scale items, it might be necessary to remove ticks at the intersections, by overloading [updateScaleDiv\(\)](#).

The scale might be at a specific position (f.e 0.0) or it might be aligned to a canvas border.

Example

The following example shows how to replace the left axis, by a scale item at the x position 0.0.

```
QwtPlotScaleItem *scaleItem =
    new QwtPlotScaleItem(QwtScaleDraw::RightScale, 0.0);
scaleItem->setFont(plot->axisWidget(QwtPlot::yLeft)->font());
scaleItem->attach(plot);

plot->enableAxis(QwtPlot::yLeft, false);
```

12.63.2 Member Enumeration Documentation

12.63.2.1 enum QwtPlotItem::ItemAttribute [inherited]

Plot Item Attributes

- Legend
The item is represented on the legend.
- AutoScale
The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.63.2.2 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.63.2.3 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.63.3 Constructor & Destructor Documentation

12.63.3.1 QwtPlotScaleItem::QwtPlotScaleItem (QwtScaleDraw::Alignment *alignment* = *QwtScaleDraw::BottomScale*, const double *pos* = *0.0*) [explicit]

Constructor for scale item at the position *pos*.

Parameters

<i>alignment</i>	In case of <i>QwtScaleDraw::BottomScale</i> / <i>QwtScaleDraw::TopScale</i> the scale item is corresponding to the xAxis() , otherwise it corresponds to the yAxis() .
<i>pos</i>	x or y position, depending on the corresponding axis.

See also

[setPosition\(\)](#), [setAlignment\(\)](#)

12.63.3.2 QwtPlotScaleItem::~~QwtPlotScaleItem () [virtual]

Destructor.

12.63.4 Member Function Documentation

12.63.4.1 void QwtPlotItem::attach (QwtPlot **plot*) [inherited]

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.63.4.2 int QwtPlotScaleItem::borderDistance () const

Returns

Distance from a canvas border

See also

[setBorderDistance\(\)](#), [setPosition\(\)](#)

12.63.4.3 QwtDoubleRect QwtPlotItem::boundingRect () const
[virtual, inherited]

Returns

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

Reimplemented in [QwtPlotCurve](#), [QwtPlotMarker](#), [QwtPlotSpectrogram](#), and [QwtPlotSvgItem](#).

12.63.4.4 void QwtPlotItem::detach () [inline, inherited]

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with.

[detach\(\)](#) is equivalent to calling [attach\(NULL \)](#)

See also

[attach\(QwtPlot* plot \)](#)

12.63.4.5 `void QwtPlotScaleItem::draw (QPainter * p, const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & rect) const`
[virtual]

Draw the scale.

Implements [QwtPlotItem](#).

12.63.4.6 `QFont QwtPlotScaleItem::font () const`

Returns

tick label font

See also

[setFont\(\)](#)

12.63.4.7 `void QwtPlotItem::hide ()` **[inherited]**

Hide the item.

12.63.4.8 `QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & rect) const`
[inherited]

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.63.4.9 bool QwtPlotScaleItem::isScaleDivFromAxis () const

Returns

True, if the synchronization of the scale division with the corresponding axis is enabled.

See also

[setScaleDiv\(\)](#), [setScaleDivFromAxis\(\)](#)

12.63.4.10 bool QwtPlotItem::isVisible () const [inherited]

Returns

true if visible

See also

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

12.63.4.11 void QwtPlotItem::itemChanged () [virtual, inherited]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.63.4.12 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.63.4.13 `QRect QwtPlotItem::paintRect (const QwtScaleMap & xMap,
const QwtScaleMap & yMap) const` `[inherited]`

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.63.4.14 `QPalette QwtPlotScaleItem::palette () const`**Returns**

palette

See also

[setPalette\(\)](#)

12.63.4.15 `QwtPlot * QwtPlotItem::plot () const` `[inherited]`

Return attached plot.

12.63.4.16 `double QwtPlotScaleItem::position () const`**Returns**

Position of the scale

See also

[setPosition\(\)](#), [setAlignment\(\)](#)

12.63.4.17 `int QwtPlotScaleItem::rtti () const [virtual]`**Returns**

QwtPlotItem::Rtti_PlotScale

Reimplemented from [QwtPlotItem](#).

12.63.4.18 `const QwtScaleDiv & QwtPlotScaleItem::scaleDiv () const`**Returns**

Scale division

12.63.4.19 `const QwtScaleDraw * QwtPlotScaleItem::scaleDraw () const`**Returns**

Scale draw

See also

[setScaleDraw\(\)](#)

12.63.4.20 `QwtScaleDraw * QwtPlotScaleItem::scaleDraw ()`**Returns**

Scale draw

See also

[setScaleDraw\(\)](#)

12.63.4.21 `QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]`

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.63.4.22 void QwtPlotScaleItem::setAlignment (QwtScaleDraw::Alignment *alignment*)

Change the alignment of the scale

The alignment sets the orientation of the scale and the position of the ticks:

- QwtScaleDraw::BottomScale: horizontal, ticks below
- QwtScaleDraw::TopScale: horizontal, ticks above
- QwtScaleDraw::LeftScale: vertical, ticks left
- QwtScaleDraw::RightScale: vertical, ticks right

For horizontal scales the position corresponds to [QwtPlotItem::yAxis\(\)](#), otherwise to [QwtPlotItem::xAxis\(\)](#).

See also

[scaleDraw\(\)](#), [QwtScaleDraw::alignment\(\)](#), [setPosition\(\)](#)

12.63.4.23 void QwtPlotItem::setAxis (int *xAxis*, int *yAxis*) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.63.4.24 void QwtPlotScaleItem::setBorderDistance (int *distance*)

Align the scale to the canvas.

If distance is ≥ 0 the scale will be aligned to a border of the contents rect of the canvas. If alignment() is QwtScaleDraw::LeftScale, the scale will be aligned to the right border, if it is QwtScaleDraw::TopScale it will be aligned to the bottom (and vice versa),

If distance is < 0 the scale will be at the [position\(\)](#).

Parameters

<i>distance</i>	Number of pixels between the canvas border and the backbone of the scale.
-----------------	---

See also

[setPosition\(\)](#), [borderDistance\(\)](#)

12.63.4.25 void QwtPlotScaleItem::setFont (const QFont & font)

Change the tick label font

See also

[font\(\)](#)

12.63.4.26 void QwtPlotItem::setItemAttribute (ItemAttribute attribute, bool on = true) [inherited]

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.63.4.27 void QwtPlotScaleItem::setPalette (const QPalette & palette)

Set the palette

See also

[QwtAbstractScaleDraw::draw\(\)](#), [palette\(\)](#)

12.63.4.28 void QwtPlotScaleItem::setPosition (double *pos*)

Change the position of the scale

The position is interpreted as y value for horizontal axes and as x value for vertical axes.

The border distance is set to -1.

Parameters

<i>pos</i>	New position
------------	--------------

See also

[position\(\)](#), [setAlignment\(\)](#)

12.63.4.29 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on* = true) [inherited]

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.63.4.30 void QwtPlotScaleItem::setScaleDiv (const QwtScaleDiv & *scaleDiv*)

Assign a scale division.

When assigning a scaleDiv the scale division won't be synchronized with the corresponding axis anymore.

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

See also

[scaleDiv\(\)](#), [setScaleDivFromAxis\(\)](#), [isScaleDivFromAxis\(\)](#)

12.63.4.31 void QwtPlotScaleItem::setScaleDivFromAxis (bool *on*)

Enable/Disable the synchronization of the scale division with the corresponding axis.

Parameters

<i>on</i>	true/false
-----------	------------

See also

[isScaleDivFromAxis\(\)](#)

12.63.4.32 void QwtPlotScaleItem::setScaleDraw (QwtScaleDraw * *scaleDraw*)

Set a scale draw.

Parameters

<i>scaleDraw</i>	object responsible for drawing scales.
------------------	--

The main use case for replacing the default [QwtScaleDraw](#) is to overload [QwtAbstractScaleDraw::label](#), to replace or swallow tick labels.

See also

[scaleDraw\(\)](#)

**12.63.4.33 void QwtPlotItem::setTitle (const QString & *title*)
[inherited]**

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

**12.63.4.34 void QwtPlotItem::setTitle (const QwtText & *title*)
[inherited]**

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also[title\(\)](#)

12.63.4.35 void QwtPlotItem::setVisible (bool *on*) [**virtual**,
inherited]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also[isVisible\(\)](#), [show\(\)](#), [hide\(\)](#)

12.63.4.36 void QwtPlotItem::setXAxis (int *axis*) [**inherited**]

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)

12.63.4.37 void QwtPlotItem::setYAxis (int *axis*) [**inherited**]

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)

12.63.4.38 void QwtPlotItem::setZ (double *z*) [inherited]

Set the *z* value.

Plot items are painted in increasing *z*-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also

[z\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.63.4.39 void QwtPlotItem::show () [inherited]

Show the item.

**12.63.4.40 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*)
const [inherited]**

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also

[setItemAttribute\(\)](#), [ItemAttribute](#)

**12.63.4.41 bool QwtPlotItem::testRenderHint (RenderHint *hint*) const
[inherited]**

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.63.4.42 `const QwtText & QwtPlotItem::title () const` **[inherited]**

Returns

Title of the item

See also

[setTitle\(\)](#)

12.63.4.43 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap,
const QwtScaleMap & yMap, const QwtDoubleRect & rect) const`
[inherited]

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.63.4.44 `void QwtPlotItem::updateLegend (QwtLegend * legend) const`
[virtual, inherited]

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also

[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

12.63.4.45 `void QwtPlotScaleItem::updateScaleDiv (const QwtScaleDiv & xScaleDiv, const QwtScaleDiv & yScaleDiv) [virtual]`

Update the item to changes of the axes scale division.

In case of [isScaleDivFromAxis\(\)](#), the scale draw is synchronized to the correspond axis.

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented from [QwtPlotItem](#).

12.63.4.46 `int QwtPlotItem::xAxis () const [inherited]`

Return xAxis.

12.63.4.47 `int QwtPlotItem::yAxis () const [inherited]`

Return yAxis.

12.63.4.48 `double QwtPlotItem::z () const [inherited]`

Plot items are painted in increasing z-order.

Returns

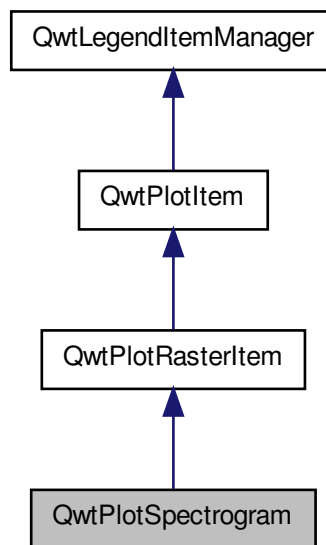
[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.64 QwtPlotSpectrogram Class Reference

A plot item, which displays a spectrogram.

```
#include <qwt_plot_spectrogram.h>
```

Inheritance diagram for QwtPlotSpectrogram:



Public Types

- enum [CachePolicy](#) {
 NoCache,
 PaintCache,
 ScreenCache }
- enum [DisplayMode](#) {
 ImageMode = 1,
 ContourMode = 2 }
- enum [ItemAttribute](#) {
 Legend = 1,
 AutoScale = 2 }
- enum [RenderHint](#) { **RenderAntialiased** = 1 }

- enum [RttiValues](#) {
Rtti_PlotItem = 0,
Rtti_PlotGrid,
Rtti_PlotScale,
Rtti_PlotMarker,
Rtti_PlotCurve,
Rtti_PlotHistogram,
Rtti_PlotSpectrogram,
Rtti_PlotSVG,
Rtti_PlotUserItem = 1000 }

Public Member Functions

- int [alpha](#) () const
- void [attach](#) ([QwtPlot](#) *plot)
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- [CachePolicy](#) [cachePolicy](#) () const
- const [QwtColorMap](#) & [colorMap](#) () const
- [QwtValueList](#) [contourLevels](#) () const
- virtual [QPen](#) [contourPen](#) (double level) const
- const [QwtRasterData](#) & [data](#) () const
- [QPen](#) [defaultContourPen](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &rect) const
- void [hide](#) ()
- void [invalidateCache](#) ()
- [QwtDoubleRect](#) [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QRect](#) &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual [QWidget](#) * [legendItem](#) () const
- [QRect](#) [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- [QwtPlot](#) * [plot](#) () const
- [QwtPlotSpectrogram](#) (const [QString](#) &title=[QString::null](#))
- virtual [QSize](#) [rasterHint](#) (const [QwtDoubleRect](#) &) const
- virtual int [rtti](#) () const
- [QwtDoubleRect](#) [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAlpha](#) (int alpha)
- void [setAxis](#) (int xAxis, int yAxis)
- void [setCachePolicy](#) ([CachePolicy](#))
- void [setColorMap](#) (const [QwtColorMap](#) &)
- void [setConrecAttribute](#) ([QwtRasterData::ConrecAttribute](#), bool on)
- void [setContourLevels](#) (const [QwtValueList](#) &)

- void [setData](#) (const [QwtRasterData](#) &data)
- void [setDefaultContourPen](#) (const [QPen](#) &)
- void [setDisplayMode](#) ([DisplayMode](#), bool on=true)
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setTitle](#) (const [QwtText](#) &title)
- void [setTitle](#) (const [QString](#) &title)
- virtual void [setVisible](#) (bool)
- void [setXAxis](#) (int axis)
- void [setYAxis](#) (int axis)
- void [setZ](#) (double z)
- void [show](#) ()
- bool [testConrecAttribute](#) ([QwtRasterData::ConrecAttribute](#)) const
- bool [testDisplayMode](#) ([DisplayMode](#)) const
- bool [testItemAttribute](#) ([ItemAttribute](#)) const
- bool [testRenderHint](#) ([RenderHint](#)) const
- const [QwtText](#) & [title](#) () const
- [QRect](#) [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QwtDoubleRect](#) &) const
- virtual void [updateLegend](#) ([QwtLegend](#) *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- int [xAxis](#) () const
- int [yAxis](#) () const
- double [z](#) () const
- virtual [~QwtPlotSpectrogram](#) ()

Protected Member Functions

- virtual [QSize](#) [contourRasterSize](#) (const [QwtDoubleRect](#) &, const [QRect](#) &) const
- virtual void [drawContourLines](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QwtRasterData::ContourLines](#) &lines) const
- virtual [QwtRasterData::ContourLines](#) [renderContourLines](#) (const [QwtDoubleRect](#) &rect, const [QSize](#) &raster) const
- virtual [QImage](#) [renderImage](#) (const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QwtDoubleRect](#) &rect) const

12.64.1 Detailed Description

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

In ContourMode contour lines are painted for the contour levels.

See also

[QwtRasterData](#), [QwtColorMap](#)

12.64.2 Member Enumeration Documentation

12.64.2.1 enum QwtPlotRasterItem::CachePolicy [inherited]

- NoCache
[renderImage\(\)](#) is called, whenever the item has to be repainted
- PaintCache
[renderImage\(\)](#) is called, whenever the image cache is not valid, or the scales, or the size of the canvas has changed. This type of cache is only useful for improving the performance of hide/show operations. All other situations are already handled by the plot canvas cache.
- ScreenCache
The screen cache is an image in size of the screen. As long as the scales don't change the target image is scaled from the cache. This might improve the performance when resizing the plot widget, but suffers from scaling effects.

The default policy is NoCache

12.64.2.2 enum QwtPlotSpectrogram::DisplayMode

The display mode controls how the raster data will be represented.

- ImageMode
The values are mapped to colors using a color map.
- ContourMode
The data is displayed using contour lines

When both modes are enabled the contour lines are painted on top of the spectrogram. The default setting enables ImageMode.

See also

[setDisplayMode\(\)](#), [testDisplayMode\(\)](#)

12.64.2.3 enum QwtPlotItem::ItemAttribute [inherited]

Plot Item Attributes

- Legend
The item is represented on the legend.

- AutoScale

The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.64.2.4 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.64.2.5 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.64.3 Constructor & Destructor Documentation

12.64.3.1 QwtPlotSpectrogram::QwtPlotSpectrogram (const QString & title = QString::null) [explicit]

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

The z value is initialized by 8.0.

Parameters

<i>title</i>	Title
--------------	-------

See also

[QwtPlotItem::setItemAttribute\(\)](#), [QwtPlotItem::setZ\(\)](#)

12.64.3.2 QwtPlotSpectrogram::~QwtPlotSpectrogram () [virtual]

Destructor.

12.64.4 Member Function Documentation

12.64.4.1 `int QwtPlotRasterItem::alpha () const` `[inherited]`

Returns

Alpha value of the raster item

See also

[setAlpha\(\)](#)

12.64.4.2 `void QwtPlotItem::attach (QwtPlot *plot)` `[inherited]`

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.64.4.3 `QwtDoubleRect QwtPlotSpectrogram::boundingRect () const` `[virtual]`

Returns

Bounding rect of the data

See also

[QwtRasterData::boundingRect\(\)](#)

Reimplemented from [QwtPlotItem](#).

12.64.4.4 `QwtPlotRasterItem::CachePolicy QwtPlotRasterItem::cachePolicy () const` `[inherited]`

Returns

Cache policy

See also

[CachePolicy](#), [setCachePolicy\(\)](#)

12.64.4.5 const QColorMap & QwtPlotSpectrogram::colorMap () const**Returns**

Color Map used for mapping the intensity values to colors

See also

[setColorMap\(\)](#)

12.64.4.6 QwtValueList QwtPlotSpectrogram::contourLevels () const

Return the levels of the contour lines.

The levels are sorted in increasing order.

See also

[contourLevels\(\)](#), [renderContourLines\(\)](#), [QwtRasterData::contourLines\(\)](#)

**12.64.4.7 QPen QwtPlotSpectrogram::contourPen (double *level*) const
[virtual]**

Calculate the pen for a contour line.

The color of the pen is the color for level calculated by the color map

Parameters

<i>level</i> Contour level

Returns

Pen for the contour line

Note

contourPen is only used if [defaultContourPen\(\).style\(\)](#) == Qt::NoPen

See also

[setDefaultContourPen\(\)](#), [setColorMap\(\)](#), [setContourLevels\(\)](#)

12.64.4.8 `QSize QwtPlotSpectrogram::contourRasterSize (const QwtDoubleRect & area, const QRect & rect) const` **[protected, virtual]**

Return the raster to be used by the CONREC contour algorithm.

A larger size will improve the precision of the CONREC algorithm, but will slow down the time that is needed to calculate the lines.

The default implementation returns `rect.size() / 2` bounded to [data\(\).rasterHint\(\)](#).

Parameters

<i>area</i>	Rect, where to calculate the contour lines
<i>rect</i>	Rect in pixel coordinates, where to paint the contour lines

Returns

Raster to be used by the CONREC contour algorithm.

Note

The size will be bounded to `rect.size()`.

See also

[drawContourLines\(\)](#), [QwtRasterData::contourLines\(\)](#)

12.64.4.9 `const QwtRasterData & QwtPlotSpectrogram::data () const`

Returns

Spectrogram data

See also

[setData\(\)](#)

12.64.4.10 `QPen QwtPlotSpectrogram::defaultContourPen () const`

Returns

Default contour pen

See also

[setDefaultContourPen\(\)](#)

12.64.4.11 void QwtPlotItem::detach () [inline, inherited]

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with.

[detach\(\)](#) is equivalent to calling `attach(NULL)`

See also

[attach\(QwtPlot* plot \)](#)

12.64.4.12 void QwtPlotSpectrogram::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the spectrogram.

Parameters

<i>painter</i>	Painter
<i>xMap</i>	Maps x-values into pixel coordinates.
<i>yMap</i>	Maps y-values into pixel coordinates.
<i>canvasRect</i>	Contents rect of the canvas in painter coordinates

See also

[setDisplayMode\(\)](#), [renderImage\(\)](#), [QwtPlotRasterItem::draw\(\)](#), [drawContourLines\(\)](#)

Reimplemented from [QwtPlotRasterItem](#).

12.64.4.13 void QwtPlotSpectrogram::drawContourLines (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QwtRasterData::ContourLines & *contourLines*) const [protected, virtual]

Paint the contour lines

Parameters

<i>painter</i>	Painter
----------------	---------

<i>xMap</i>	Maps x-values into pixel coordinates.
<i>yMap</i>	Maps y-values into pixel coordinates.
<i>contourLines</i>	Contour lines

See also

[renderContourLines\(\)](#), [defaultContourPen\(\)](#), [contourPen\(\)](#)

12.64.4.14 void QwtPlotItem::hide () [inherited]

Hide the item.

12.64.4.15 void QwtPlotRasterItem::invalidateCache () [inherited]

Invalidate the paint cache

See also

[setCachePolicy\(\)](#)

12.64.4.16 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRect & rect) const [inherited]

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.64.4.17 bool QwtPlotItem::isVisible () const [inherited]

Returns

true if visible

See also

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

12.64.4.18 void QwtPlotItem::itemChanged () [virtual, inherited]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.64.4.19 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.64.4.20 QRect QwtPlotItem::paintRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.64.4.21 QwtPlot * QwtPlotItem::plot () const [inherited]

Return attached plot.

12.64.4.22 QSize QwtPlotSpectrogram::rasterHint (const QwtDoubleRect & rect) const [virtual]

Returns the recommended raster for a given rect.

E.g. the raster hint is used to limit the resolution of the image that is rendered.

Parameters

<i>rect</i>	Rect for the raster hint
-------------	--------------------------

Returns

[data\(\)](#).rasterHint(rect)

Reimplemented from [QwtPlotRasterItem](#).

12.64.4.23 QwtRasterData::ContourLines QwtPlotSpectrogram::renderContourLines (const QwtDoubleRect & rect, const QSize & raster) const [protected, virtual]

Calculate contour lines

Parameters

<i>rect</i>	Rectangle, where to calculate the contour lines
<i>raster</i>	Raster, used by the CONREC algorithm

See also

[contourLevels\(\)](#), [setConrecAttribute\(\)](#), [QwtRasterData::contourLines\(\)](#)

12.64.4.24 QImage QwtPlotSpectrogram::renderImage (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QwtDoubleRect & area) 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

<i>xMap</i>	X-Scale Map
<i>yMap</i>	Y-Scale Map
<i>area</i>	Area that should be rendered in scale coordinates.

Returns

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

See also

QwtRasterData::intensity(), [QwtColorMap::rgb\(\)](#), [QwtColorMap::colorIndex\(\)](#)

Implements [QwtPlotRasterItem](#).

12.64.4.25 int QwtPlotSpectrogram::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotSpectrogram

Reimplemented from [QwtPlotItem](#).

12.64.4.26 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & xMap, const QwtScaleMap & yMap) const [inherited]

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.64.4.27 void QwtPlotRasterItem::setAlpha (int alpha) [inherited]

Set an alpha value for the raster data.

Often a plot has several types of raster data organized in layers. (f.e a geographical map, with weather statistics). Using [setAlpha\(\)](#) raster items can be stacked easily.

The alpha value is a value [0, 255] to control the transparency of the image. 0 represents a fully transparent color, while 255 represents a fully opaque color.

Parameters

<i>alpha</i>	Alpha value
--------------	-------------

- $\alpha \geq 0$
All alpha values of the pixels returned by [renderImage\(\)](#) will be set to alpha, beside those with an alpha value of 0 (invalid pixels).
- $\alpha < 0$ The alpha values returned by [renderImage\(\)](#) are not changed.

The default alpha value is -1.

See also

[alpha\(\)](#)

12.64.4.28 void QwtPlotItem::setAxis (int xAxis, int yAxis) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.64.4.29 void QwtPlotRasterItem::setCachePolicy (QwtPlotRasterItem::CachePolicy policy) [inherited]

Change the cache policy

The default policy is NoCache

Parameters

<i>policy</i>	Cache policy
---------------	--------------

See also

[CachePolicy](#), [cachePolicy\(\)](#)

12.64.4.30 void QwtPlotSpectrogram::setColorMap (const 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

<i>colorMap</i>	Color Map
-----------------	-----------

See also

[colorMap\(\)](#), [QwtScaleWidget::setColorBarEnabled\(\)](#), [QwtScaleWidget::setColorMap\(\)](#)

12.64.4.31 void QwtPlotSpectrogram::setConrecAttribute (QwtRasterData::ConrecAttribute *attribute*, bool *on*)

Modify an attribute of the CONREC algorithm, used to calculate the contour lines.

Parameters

<i>attribute</i>	CONREC attribute
<i>on</i>	On/Off

See also

[testConrecAttribute\(\)](#), [renderContourLines\(\)](#), [QwtRasterData::contourLines\(\)](#)

12.64.4.32 void QwtPlotSpectrogram::setContourLevels (const QwtValueList & *levels*)

Set the levels of the contour lines

Parameters

<i>levels</i>	Values of the contour levels
---------------	------------------------------

See also

[contourLevels\(\)](#), [renderContourLines\(\)](#), [QwtRasterData::contourLines\(\)](#)

Note

[contourLevels](#) returns the same levels but sorted.

12.64.4.33 void QwtPlotSpectrogram::setData (const QwtRasterData & *data*)

Set the data to be displayed

Parameters

<i>data</i>	Spectrogram Data
-------------	------------------

See also[data\(\)](#)**12.64.4.34 void QwtPlotSpectrogram::setDefaultContourPen (const QPen & *pen*)**

Set the default pen for the contour lines.

If the spectrogram has a valid default contour pen a contour line is painted using the default contour pen. Otherwise (`pen.style() == Qt::NoPen`) the pen is calculated for each contour level using [contourPen\(\)](#).

See also[defaultContourPen\(\)](#), [contourPen\(\)](#)**12.64.4.35 void QwtPlotSpectrogram::setDisplayMode (DisplayMode *mode*, bool *on* = *true*)**

The display mode controls how the raster data will be represented.

Parameters

<i>mode</i>	Display mode
<i>on</i>	On/Off

The default setting enables ImageMode.

See also[DisplayMode](#), [displayMode\(\)](#)**12.64.4.36 void QwtPlotItem::setItemAttribute (ItemAttribute *attribute*, bool *on* = *true*) [inherited]**

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also[testItemAttribute\(\)](#), [ItemAttribute](#)

12.64.4.37 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on* = **true**) [**inherited**]

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also[testRenderHint\(\)](#), [RenderHint](#)

12.64.4.38 void QwtPlotItem::setTitle (const QwtText & *title*) [**inherited**]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also[title\(\)](#)

12.64.4.39 void QwtPlotItem::setTitle (const QString & *title*) [**inherited**]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also[title\(\)](#)

12.64.4.40 void QwtPlotItem::setVisible (bool *on*) [virtual, inherited]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also[isVisible\(\)](#), [show\(\)](#), [hide\(\)](#)**12.64.4.41 void QwtPlotItem::setXAxis (int *axis*) [inherited]**

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)**12.64.4.42 void QwtPlotItem::setYAxis (int *axis*) [inherited]**

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)**12.64.4.43 void QwtPlotItem::setZ (double *z*) [inherited]**

Set the z value.

Plot items are painted in increasing z-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also

[z\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.64.4.44 void QwtPlotItem::show () [inherited]

Show the item.

12.64.4.45 bool QwtPlotSpectrogram::testConrecAttribute (QwtRasterData::ConrecAttribute *attribute*) const

Test an attribute of the CONREC algorithm, used to calculate the contour lines.

Parameters

<i>attribute</i>	CONREC attribute
------------------	------------------

Returns

true, is enabled

See also

[setConrecAttribute\(\)](#), [renderContourLines\(\)](#), [QwtRasterData::contourLines\(\)](#)

12.64.4.46 bool QwtPlotSpectrogram::testDisplayMode (DisplayMode *mode*) const

The display mode controls how the raster data will be represented.

Parameters

<i>mode</i>	Display mode
-------------	--------------

Returns

true if mode is enabled

12.64.4.47 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*) const [inherited]

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also[setItemAttribute\(\)](#), [ItemAttribute](#)

12.64.4.48 `bool QwtPlotItem::testRenderHint (RenderHint hint) const`
[inherited]

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also[setRenderHint\(\)](#), [RenderHint](#)

12.64.4.49 `const QwtText & QwtPlotItem::title () const` **[inherited]**

Returns

Title of the item

See also[setTitle\(\)](#)

12.64.4.50 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap,
const QwtScaleMap & yMap, const QwtDoubleRect & rect) const`
[inherited]

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.64.4.51 `void QwtPlotItem::updateLegend (QwtLegend * legend) const`
[virtual, inherited]

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also

[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

12.64.4.52 `void QwtPlotItem::updateScaleDiv (const QwtScaleDiv & , const QwtScaleDiv &) [virtual, inherited]`

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 [QwtPlotGrid\(\)](#)) have to reimplement [updateScaleDiv\(\)](#)

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented in [QwtPlotGrid](#), and [QwtPlotScaleItem](#).

12.64.4.53 `int QwtPlotItem::xAxis () const [inherited]`

Return xAxis.

12.64.4.54 `int QwtPlotItem::yAxis () const [inherited]`

Return yAxis.

12.64.4.55 `double QwtPlotItem::z () const [inherited]`

Plot items are painted in increasing z-order.

Returns

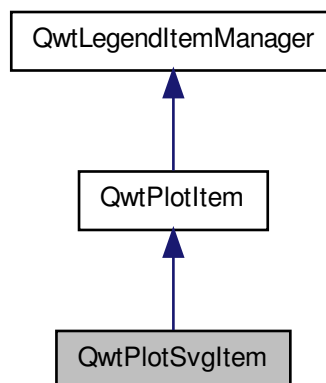
[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.65 QwtPlotSvgItem Class Reference

A plot item, which displays data in Scalable Vector Graphics (SVG) format.

```
#include <qwt_plot_svgitem.h>
```

Inheritance diagram for QwtPlotSvgItem:



Public Types

- enum [ItemAttribute](#) {
Legend = 1,
AutoScale = 2 }
- enum [RenderHint](#) { **RenderAntialiased** = 1 }
- enum [RttiValues](#) {
Rtti_PlotItem = 0,
Rtti_PlotGrid,
Rtti_PlotScale,
Rtti_PlotMarker,
Rtti_PlotCurve,
Rtti_PlotHistogram,
Rtti_PlotSpectrogram,
Rtti_PlotSVG,
Rtti_PlotUserItem = 1000 }

Public Member Functions

- void [attach](#) ([QwtPlot](#) *plot)
- virtual [QwtDoubleRect](#) [boundingRect](#) () const
- void [detach](#) ()
- virtual void [draw](#) ([QPainter](#) *p, const [QwtScaleMap](#) &xMap, const [QwtScaleMap](#) &yMap, const [QRect](#) &rect) const
- void [hide](#) ()
- [QwtDoubleRect](#) [invTransform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const [QRect](#) &) const
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual [QWidget](#) * [legendItem](#) () const
- bool [loadData](#) (const [QwtDoubleRect](#) &, const [QByteArray](#) &)
- bool [loadFile](#) (const [QwtDoubleRect](#) &, const [QString](#) &fileName)
- [QRect](#) [paintRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- [QwtPlot](#) * [plot](#) () const
- [QwtPlotSvgItem](#) (const [QwtText](#) &title)
- [QwtPlotSvgItem](#) (const [QString](#) &title=[QString::null](#))
- virtual int [rtti](#) () const
- [QwtDoubleRect](#) [scaleRect](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &) const
- void [setAxis](#) (int xAxis, int yAxis)
- void [setItemAttribute](#) ([ItemAttribute](#), bool on=true)
- void [setRenderHint](#) ([RenderHint](#), bool on=true)
- void [setTitle](#) (const [QString](#) &title)
- void [setTitle](#) (const [QwtText](#) &title)
- virtual void [setVisible](#) (bool)

- void [setXAxis](#) (int axis)
- void [setYAxis](#) (int axis)
- void [setZ](#) (double z)
- void [show](#) ()
- bool [testItemAttribute](#) (ItemAttribute) const
- bool [testRenderHint](#) (RenderHint) const
- const [QwtText](#) & [title](#) () const
- QRect [transform](#) (const [QwtScaleMap](#) &, const [QwtScaleMap](#) &, const QwtDoubleRect &) const
- virtual void [updateLegend](#) (QwtLegend *) const
- virtual void [updateScaleDiv](#) (const [QwtScaleDiv](#) &, const [QwtScaleDiv](#) &)
- int [xAxis](#) () const
- int [yAxis](#) () const
- double [z](#) () const
- virtual [~QwtPlotSvgItem](#) ()

Protected Member Functions

- void [render](#) (QPainter *painter, const QwtDoubleRect &viewBox, const QRect &rect) const
- QwtDoubleRect [viewBox](#) (const QwtDoubleRect &area) const

12.65.1 Detailed Description

A plot item, which displays data in Scalable Vector Graphics (SVG) format. SVG images are often used to display maps

12.65.2 Member Enumeration Documentation

12.65.2.1 enum QwtPlotItem::ItemAttribute [inherited]

Plot Item Attributes

- Legend
The item is represented on the legend.
- AutoScale
The [boundingRect\(\)](#) of the item is included in the autoscaling calculation.

See also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

12.65.2.2 enum QwtPlotItem::RenderHint [inherited]

Render hints.

12.65.2.3 enum QwtPlotItem::RttiValues [inherited]

Runtime type information.

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

12.65.3 Constructor & Destructor Documentation

12.65.3.1 QwtPlotSvgItem::QwtPlotSvgItem (const QString & title = QString::null) [explicit]

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

<i>title</i>	Title
--------------	-------

12.65.3.2 QwtPlotSvgItem::QwtPlotSvgItem (const QwtText & title) [explicit]

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

<i>title</i>	Title
--------------	-------

12.65.3.3 QwtPlotSvgItem::~~QwtPlotSvgItem () [virtual]

Destructor.

12.65.4 Member Function Documentation**12.65.4.1 void QwtPlotItem::attach (QwtPlot * *plot*) [inherited]**

Attach the item to a plot.

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

Parameters

<i>plot</i>	Plot widget
-------------	-------------

See also

[QwtPlotItem::detach\(\)](#)

12.65.4.2 QwtDoubleRect QwtPlotSvgItem::boundingRect () const [virtual]

Bounding rect of the item.

Reimplemented from [QwtPlotItem](#).

12.65.4.3 void QwtPlotItem::detach () [inline, inherited]

This method detaches a [QwtPlotItem](#) from any [QwtPlot](#) it has been associated with.

[detach\(\)](#) is equivalent to calling `attach(NULL)`

See also

[attach\(QwtPlot* plot \)](#)

12.65.4.4 void QwtPlotSvgItem::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the SVG item

Parameters

<i>painter</i>	Painter
<i>xMap</i>	X-Scale Map
<i>yMap</i>	Y-Scale Map
<i>canvasRect</i>	Contents rect of the plot canvas

Implements [QwtPlotItem](#).

12.65.4.5 void QwtPlotItem::hide () [inherited]

Hide the item.

12.65.4.6 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *rect*) const [inherited]

Transform a rectangle from paint to scale coordinates

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

[transform\(\)](#)

12.65.4.7 bool QwtPlotItem::isVisible () const [inherited]**Returns**

true if visible

See also

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

12.65.4.8 void QwtPlotItem::itemChanged () [virtual, inherited]

Update the legend and call [QwtPlot::autoRefresh](#) for the parent plot.

See also

[updateLegend\(\)](#)

12.65.4.9 QWidget * QwtPlotItem::legendItem () const [virtual, inherited]

Allocate the widget that represents the item on the legend.

The default implementation is made for [QwtPlotCurve](#) and returns a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Returns

[QwtLegendItem\(\)](#)

See also

[updateLegend\(\)](#) [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

12.65.4.10 bool QwtPlotSvgItem::loadData (const QwtDoubleRect & rect, const QByteArray & data)

Load SVG data

Parameters

<i>rect</i>	Bounding rectangle
<i>data</i>	in SVG format

Returns

true, if the SVG data could be loaded

12.65.4.11 bool QwtPlotSvgItem::loadFile (const QwtDoubleRect & rect, const QString & fileName)

Load a SVG file

Parameters

<i>rect</i>	Bounding rectangle
<i>fileName</i>	SVG file name

Returns

true, if the SVG file could be loaded

12.65.4.12 `QRect QwtPlotItem::paintRect (const QwtScaleMap & xMap,
const QwtScaleMap & yMap) const` **[inherited]**

Calculate the bounding paint rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.65.4.13 `QwtPlot * QwtPlotItem::plot () const` **[inherited]**

Return attached plot.

12.65.4.14 `void QwtPlotSvgItem::render (QPainter * painter, const
QwtDoubleRect & viewBox, const QRect & rect) const`
[protected]

Render the SVG data

Parameters

<i>painter</i>	Painter
<i>viewBox</i>	View Box, see QSvgRenderer::viewBox
<i>rect</i>	Traget rectangle on the paint device

12.65.4.15 `int QwtPlotSvgItem::rtti () const` **[virtual]**
Returns

QwtPlotItem::Rtti_PlotSVG

Reimplemented from [QwtPlotItem](#).

12.65.4.16 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const [inherited]

Calculate the bounding scale rect of 2 maps.

Parameters

<i>xMap</i>	X map
<i>yMap</i>	X map

Returns

Bounding rect of the scale maps

12.65.4.17 void QwtPlotItem::setAxis (int *xAxis*, int *yAxis*) [inherited]

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>xAxis</i>	X Axis
<i>yAxis</i>	Y Axis

See also

[setXAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#), [yAxis\(\)](#)

12.65.4.18 void QwtPlotItem::setItemAttribute (ItemAttribute *attribute*, bool *on = true*) [inherited]

Toggle an item attribute

Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

See also

[testItemAttribute\(\)](#), [ItemAttribute](#)

12.65.4.19 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on = true*) [inherited]

Toggle an render hint

Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

See also

[testRenderHint\(\)](#), [RenderHint](#)

12.65.4.20 void QwtPlotItem::setTitle (const QString & *title*)
[**inherited**]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.65.4.21 void QwtPlotItem::setTitle (const QwtText & *title*)
[**inherited**]

Set a new title

Parameters

<i>title</i>	Title
--------------	-------

See also

[title\(\)](#)

12.65.4.22 void QwtPlotItem::setVisible (bool *on*) [**virtual**,
inherited]

Show/Hide the item

Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

See also

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

12.65.4.23 void QwtPlotItem::setXAxis (int *axis*) [inherited]

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	X Axis
-------------	--------

See also[setAxis\(\)](#), [setYAxis\(\)](#), [xAxis\(\)](#)**12.65.4.24 void QwtPlotItem::setYAxis (int *axis*) [inherited]**

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

<i>axis</i>	Y Axis
-------------	--------

See also[setAxis\(\)](#), [setXAxis\(\)](#), [yAxis\(\)](#)**12.65.4.25 void QwtPlotItem::setZ (double *z*) [inherited]**

Set the z value.

Plot items are painted in increasing z-order.

Parameters

<i>z</i>	Z-value
----------	---------

See also[z\(\)](#), [QwtPlotDict::itemList\(\)](#)**12.65.4.26 void QwtPlotItem::show () [inherited]**

Show the item.

12.65.4.27 `bool QwtPlotItem::testItemAttribute (ItemAttribute attribute) const` `[inherited]`

Test an item attribute

Parameters

<i>attribute</i>	Attribute type
------------------	----------------

Returns

true/false

See also

[setItemAttribute\(\)](#), [ItemAttribute](#)

12.65.4.28 `bool QwtPlotItem::testRenderHint (RenderHint hint) const` `[inherited]`

Test a render hint

Parameters

<i>hint</i>	Render hint
-------------	-------------

Returns

true/false

See also

[setRenderHint\(\)](#), [RenderHint](#)

12.65.4.29 `const QwtText & QwtPlotItem::title () const` `[inherited]`

Returns

Title of the item

See also

[setTitle\(\)](#)

12.65.4.30 `QRect QwtPlotItem::transform (const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QwtDoubleRect & rect) const` `[inherited]`

Transform a rectangle

Parameters

<i>xMap</i>	X map
<i>yMap</i>	Y map
<i>rect</i>	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

[invTransform\(\)](#)

12.65.4.31 void QwtPlotItem::updateLegend (QwtLegend * *legend*) const [virtual, inherited]

Update the widget that represents the item on the legend.

[updateLegend\(\)](#) is called from [itemChanged\(\)](#) to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for [QwtPlotCurve](#) and updates a [QwtLegendItem\(\)](#), but an item could be represented by any type of widget, by overloading [legendItem\(\)](#) and [updateLegend\(\)](#).

Parameters

<i>legend</i>	Legend
---------------	--------

See also

[legendItem\(\)](#), [itemChanged\(\)](#), [QwtLegend\(\)](#)

Implements [QwtLegendItemManager](#).

Reimplemented in [QwtPlotCurve](#).

12.65.4.32 void QwtPlotItem::updateScaleDiv (const QwtScaleDiv & , const QwtScaleDiv &) [virtual, inherited]

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 [QwtPlotGrid\(\)](#)) have to reimplement [updateScaleDiv\(\)](#)

Parameters

<i>xScaleDiv</i>	Scale division of the x-axis
<i>yScaleDiv</i>	Scale division of the y-axis

See also

[QwtPlot::updateAxes\(\)](#)

Reimplemented in [QwtPlotGrid](#), and [QwtPlotScaleItem](#).

12.65.4.33 QwtDoubleRect QwtPlotSvgItem::viewBox (const QwtDoubleRect & rect) const **[protected]**

Calculate the viewBox from an rect and [boundingRect\(\)](#).

Parameters

<i>rect</i>	Rectangle in scale coordinates
-------------	--------------------------------

Returns

viewBox View Box, see [QSvgRenderer::viewBox](#)

12.65.4.34 int QwtPlotItem::xAxis () const **[inherited]**

Return xAxis.

12.65.4.35 int QwtPlotItem::yAxis () const **[inherited]**

Return yAxis.

12.65.4.36 double QwtPlotItem::z () const **[inherited]**

Plot items are painted in increasing z-order.

Returns

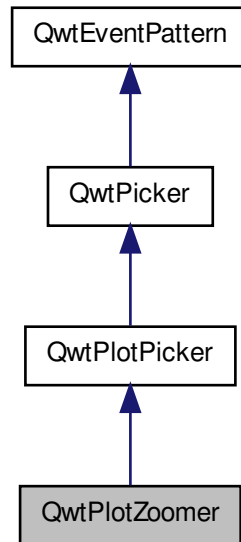
[setZ\(\)](#), [QwtPlotDict::itemList\(\)](#)

12.66 QwtPlotZoomer Class Reference

[QwtPlotZoomer](#) provides stacked zooming for a plot widget.

```
#include <qwt_plot_zoomer.h>
```


Inheritance diagram for QwtPlotZoomer:



Public Types

- enum [DisplayMode](#) {
 AlwaysOff,
 AlwaysOn,
 ActiveOnly }
- enum [KeyPatternCode](#) {
 KeySelect1,
 KeySelect2,
 KeyAbort,
 KeyLeft,
 KeyRight,
 KeyUp,
 KeyDown,
 KeyRedo,
 KeyUndo,
 KeyHome,
 KeyPatternCount }

- enum [MousePatternCode](#) {
 MouseSelect1,
 MouseSelect2,
 MouseSelect3,
 MouseSelect4,
 MouseSelect5,
 MouseSelect6,
 MousePatternCount }
- enum [RectSelectionType](#) {
 CornerToCorner = 64,
 CenterToCorner = 128,
 CenterToRadius = 256 }
- enum [ResizeMode](#) {
 Stretch,
 KeepSize }
- enum [RubberBand](#) {
 NoRubberBand = 0,
 HLineRubberBand,
 VLineRubberBand,
 CrossRubberBand,
 RectRubberBand,
 EllipseRubberBand,
 PolygonRubberBand,
 UserRubberBand = 100 }
- enum [SelectionMode](#) {
 ClickSelection = 1024,
 DragSelection = 2048 }
- enum [SelectionType](#) {
 NoSelection = 0,
 PointSelection = 1,
 RectSelection = 2,
 PolygonSelection = 4 }

Public Slots

- virtual void [move](#) (double x, double y)
- void [moveBy](#) (double x, double y)
- virtual void [zoom](#) (const QwtDoubleRect &)
- virtual void [zoom](#) (int up)

Signals

- void [appended](#) (const QwtDoublePoint &pos)
- void [appended](#) (const QPoint &pos)
- void [changed](#) (const QwtPolygon &pa)
- void [moved](#) (const QPoint &pos)
- void [moved](#) (const QwtDoublePoint &pos)
- void [selected](#) (const QwtDoubleRect &rect)
- void [selected](#) (const QwtDoublePoint &pos)
- void [selected](#) (const QwtArray< QwtDoublePoint > &pa)
- void [selected](#) (const QwtPolygon &pa)
- void [zoomed](#) (const QwtDoubleRect &rect)

Public Member Functions

- [QwtPlotCanvas * canvas](#) ()
- const [QwtPlotCanvas * canvas](#) () const
- virtual void [drawRubberBand](#) (QPainter *) const
- virtual void [drawTracker](#) (QPainter *) const
- virtual bool [eventFilter](#) (QObject *, QEvent *)
- void [initKeyPattern](#) ()
- void [initMousePattern](#) (int numButtons)
- bool [isActive](#) () const
- bool [isEnabled](#) () const
- bool [keyMatch](#) (uint pattern, const QKeyEvent *) const
- const QwtArray< [KeyPattern](#) > & [keyPattern](#) () const
- QwtArray< [KeyPattern](#) > & [keyPattern](#) ()
- int [maxStackDepth](#) () const
- bool [mouseMatch](#) (uint pattern, const QMouseEvent *) const
- const QwtArray< [MousePattern](#) > & [mousePattern](#) () const
- QwtArray< [MousePattern](#) > & [mousePattern](#) ()
- QWidget * [parentWidget](#) ()
- const QWidget * [parentWidget](#) () const
- virtual QRect [pickRect](#) () const
- const [QwtPlot * plot](#) () const
- [QwtPlot * plot](#) ()
- [QwtPlotZoomer](#) (QwtPlotCanvas *, bool doReplot=true)
- [QwtPlotZoomer](#) (int xAxis, int yAxis, QwtPlotCanvas *, bool doReplot=true)
- [QwtPlotZoomer](#) (int xAxis, int yAxis, int selectionFlags, [DisplayMode](#) tracker-Mode, QwtPlotCanvas *, bool doReplot=true)
- [ResizeMode resizeMode](#) () const
- [RubberBand rubberBand](#) () const
- QPen [rubberBandPen](#) () const
- const QwtPolygon & [selection](#) () const
- int [selectionFlags](#) () const
- virtual void [setAxis](#) (int xAxis, int yAxis)

- virtual void [setEnabled](#) (bool)
- void [setKeyPattern](#) (uint pattern, int key, int state=Qt::NoButton)
- void [setKeyPattern](#) (const QwtArray< [KeyPattern](#) > &)
- void [setMaxStackDepth](#) (int)
- void [setMousePattern](#) (const QwtArray< [MousePattern](#) > &)
- void [setMousePattern](#) (uint pattern, int button, int state=Qt::NoButton)
- virtual void [setResizeMode](#) ([ResizeMode](#))
- virtual void [setRubberBand](#) ([RubberBand](#))
- virtual void [setRubberBandPen](#) (const QPen &)
- virtual void [setSelectionFlags](#) (int)
- virtual void [setTrackerFont](#) (const QFont &)
- virtual void [setTrackerMode](#) ([DisplayMode](#))
- virtual void [setTrackerPen](#) (const QPen &)
- virtual void [setZoomBase](#) (bool doReplot=true)
- virtual void [setZoomBase](#) (const QwtDoubleRect &)
- void [setZoomStack](#) (const QStack< QwtDoubleRect > &, int zoomRectIndex=-1)
- QFont [trackerFont](#) () const
- [DisplayMode](#) [trackerMode](#) () const
- QPen [trackerPen](#) () const
- QPoint [trackerPosition](#) () const
- QRect [trackerRect](#) (const QFont &) const
- int [xAxis](#) () const
- int [yAxis](#) () const
- QwtDoubleRect [zoomBase](#) () const
- QwtDoubleRect [zoomRect](#) () const
- uint [zoomRectIndex](#) () const
- const QStack< QwtDoubleRect > & [zoomStack](#) () const

Protected Member Functions

- virtual bool [accept](#) (QwtPolygon &) const
- virtual void [append](#) (const QPoint &)
- virtual void [begin](#) ()
- virtual bool [end](#) (bool ok=true)
- QwtDoubleRect [invTransform](#) (const QRect &) const
- QwtDoublePoint [invTransform](#) (const QPoint &) const
- virtual bool [keyMatch](#) (const [KeyPattern](#) &, const QKeyEvent *) const
- virtual QwtDoubleSize [minZoomSize](#) () const
- virtual bool [mouseMatch](#) (const [MousePattern](#) &, const QMouseEvent *) const
- virtual void [move](#) (const QPoint &)
- virtual void [rescale](#) ()
- virtual void [reset](#) ()
- const QWidget * [rubberBandWidget](#) () const
- QwtDoubleRect [scaleRect](#) () const
- virtual QwtPickerMachine * [stateMachine](#) (int) const

- virtual void [stretchSelection](#) (const QSize &oldSize, const QSize &newSize)
- virtual [QwtText trackerText](#) (const QwtDoublePoint &) const
- virtual [QwtText trackerText](#) (const QPoint &) const
- const QWidget * [trackerWidget](#) () const
- QRect [transform](#) (const QwtDoubleRect &) const
- QPoint [transform](#) (const QwtDoublePoint &) const
- virtual void [transition](#) (const QEvent *)
- virtual void [updateDisplay](#) ()
- virtual void [widgetKeyPressEvent](#) (QKeyEvent *)
- virtual void [widgetKeyReleaseEvent](#) (QKeyEvent *)
- virtual void [widgetLeaveEvent](#) (QEvent *)
- virtual void [widgetMouseDoubleClickEvent](#) (QMouseEvent *)
- virtual void [widgetMouseMoveEvent](#) (QMouseEvent *)
- virtual void [widgetMousePressEvent](#) (QMouseEvent *)
- virtual void [widgetMouseReleaseEvent](#) (QMouseEvent *)
- virtual void [widgetWheelEvent](#) (QWheelEvent *)

12.66.1 Detailed Description

[QwtPlotZoomer](#) provides stacked zooming for a plot widget. [QwtPlotZoomer](#) offers rubberband selections on the plot canvas, translating the selected rectangles into plot coordinates and adjusting the axes to them. Zooming can be repeated as often as possible, limited only by [maxStackDepth\(\)](#) or [minZoomSize\(\)](#). Each rectangle is pushed on a stack.

Zoom rectangles can be selected depending on [selectionFlags\(\)](#) using the mouse or keyboard ([QwtEventPattern](#), [QwtPickerMachine](#)). [QwtEventPattern::MouseSelect3](#)/[QwtEventPattern::KeyUndo](#), or [QwtEventPattern::MouseSelect6](#)/[QwtEventPattern::KeyRedo](#) walk up and down the zoom stack. [QwtEventPattern::MouseSelect2](#) or [QwtEventPattern::KeyHome](#) unzoom to the initial size.

[QwtPlotZoomer](#) is tailored for plots with one x and y axis, but it is allowed to attach a second [QwtPlotZoomer](#) for the other axes.

Note

The realtime example includes an derived zoomer class that adds scrollbars to the plot canvas.

12.66.2 Member Enumeration Documentation

12.66.2.1 enum QwtPicker::DisplayMode [inherited]

- AlwaysOff
Display never.

- AlwaysOn
Display always.
- ActiveOnly
Display only when the selection is active.

See also

[QwtPicker::setTrackerMode\(\)](#), [QwtPicker::trackerMode\(\)](#), [QwtPicker::isActive\(\)](#)

12.66.2.2 enum QwtEventPattern::KeyPatternCode [inherited]

Symbolic keyboard input codes.

Default initialization:

- KeySelect1
Qt::Key_Return
- KeySelect2
Qt::Key_Space
- KeyAbort
Qt::Key_Escape
- KeyLeft
Qt::Key_Left
- KeyRight
Qt::Key_Right
- KeyUp
Qt::Key_Up
- KeyDown
Qt::Key_Down
- KeyUndo
Qt::Key_Minus
- KeyRedo
Qt::Key_Plus
- KeyHome
Qt::Key_Escape

12.66.2.3 enum QwtEventPattern::MousePatternCode [inherited]

Symbolic mouse input codes.

The default initialization for 3 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::MidButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::MidButton + Qt::ShiftButton

The default initialization for 2 button mice is:

- MouseSelect1
Qt::LeftButton
- MouseSelect2
Qt::RightButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::RightButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

The default initialization for 1 button mice is:

- MouseSelect1
Qt::LeftButton

- MouseSelect2
Qt::LeftButton + Qt::ControlButton
- MouseSelect3
Qt::LeftButton + Qt::AltButton
- MouseSelect4
Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
Qt::LeftButton + Qt::ControlButton + Qt::ShiftButton
- MouseSelect6
Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

See also

[initMousePattern\(\)](#)

12.66.2.4 enum QwtPicker::RectSelectionType [inherited]

Selection subtype for RectSelection This enum type describes the type of rectangle selections. It can be or'd with [QwtPicker::RectSelectionType](#) and [QwtPicker::SelectionMode](#) and passed to [QwtPicker::setSelectionFlags\(\)](#).

- CornerToCorner
The first and the second selected point are the corners of the rectangle.
- CenterToCorner
The first point is the center, the second a corner of the rectangle.
- CenterToRadius
The first point is the center of a quadrat, calculated by the maximum of the x- and y-distance.

The default value is CornerToCorner.

See also

[QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::selectionFlags\(\)](#)

12.66.2.5 enum QwtPicker::ResizeMode [inherited]

Controls what to do with the selected points of an active selection when the observed widget is resized.

- Stretch
All points are scaled according to the new size,
- KeepSize
All points remain unchanged.

The default value is Stretch.

See also

[QwtPicker::setResizeMode\(\)](#), [QwtPicker::resize\(\)](#)

12.66.2.6 enum QwtPicker::RubberBand [inherited]

Rubberband style

- NoRubberBand
No rubberband.
- HLineRubberBand & PointSelection
A horizontal line.
- VLineRubberBand & PointSelection
A vertical line.
- CrossRubberBand & PointSelection
A horizontal and a vertical line.
- RectRubberBand & RectSelection
A rectangle.
- EllipseRubberBand & RectSelection
An ellipse.
- PolygonRubberBand & PolygonSelection
A polygon.
- UserRubberBand
Values \geq UserRubberBand can be used to define additional rubber bands.

The default value is NoRubberBand.

See also

[QwtPicker::setRubberBand\(\)](#), [QwtPicker::rubberBand\(\)](#)

12.66.2.7 enum QwtPicker::SelectionMode [inherited]

Values of this enum type or'd together with a SelectionType value identifies which state machine should be used for the selection.

The default value is ClickSelection.

See also

[stateMachine\(\)](#)

12.66.2.8 enum QwtPicker::SelectionType [inherited]

This enum type describes the type of a selection. It can be or'd with [QwtPicker::RectSelectionType](#) and [QwtPicker::SelectionMode](#) and passed to [QwtPicker::setSelectionFlags\(\)](#)

- NoSelection
Selection is disabled. Note this is different to the disabled state, as you might have a tracker.
- PointSelection
Select a single point.
- RectSelection
Select a rectangle.
- PolygonSelection
Select a polygon.

The default value is NoSelection.

See also

[QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::selectionFlags\(\)](#)

12.66.3 Constructor & Destructor Documentation

12.66.3.1 QwtPlotZoomer::QwtPlotZoomer (QwtPlotCanvas * *canvas*, bool *doReplot* = *true*) [explicit]

Create a zoomer for a plot canvas.

The zoomer is set to those x- and y-axis of the parent plot of the canvas that are enabled. If both or no x-axis are enabled, the picker is set to QwtPlot::xBottom. If both or no y-axis are enabled, it is set to QwtPlot::yLeft.

The [selectionFlags\(\)](#) are set to QwtPicker::RectSelection | QwtPicker::ClickSelection, the tracker mode to QwtPicker::ActiveOnly.

Parameters

<i>canvas</i>	Plot canvas to observe, also the parent object
<i>doReplot</i>	Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#), [setZoomBase\(\)](#)

12.66.3.2 QwtPlotZoomer::QwtPlotZoomer (int *xAxis*, int *yAxis*, QwtPlotCanvas * *canvas*, bool *doReplot* = *true*) [explicit]

Create a zoomer for a plot canvas.

The [selectionFlags\(\)](#) are set to `QwtPicker::RectSelection | QwtPicker::ClickSelection`, the tracker mode to `QwtPicker::ActiveOnly`.

Parameters

<i>xAxis</i>	X axis of the zoomer
<i>yAxis</i>	Y axis of the zoomer
<i>canvas</i>	Plot canvas to observe, also the parent object
<i>doReplot</i>	Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#), [setZoomBase\(\)](#)

12.66.3.3 QwtPlotZoomer::QwtPlotZoomer (int *xAxis*, int *yAxis*, int *selectionFlags*, *DisplayMode* *trackerMode*, QwtPlotCanvas * *canvas*, bool *doReplot* = *true*) [explicit]

Create a zoomer for a plot canvas.

Parameters

<i>xAxis</i>	X axis of the zoomer
<i>yAxis</i>	Y axis of the zoomer
<i>selection-Flags</i>	Or'd value of QwtPicker::RectSelectionType and QwtPicker::SelectionMode . <code>QwtPicker::RectSelection</code> will be auto added.
<i>trackerMode</i>	Tracker mode
<i>canvas</i>	Plot canvas to observe, also the parent object
<i>doReplot</i>	Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

[QwtPicker](#), [QwtPicker::setSelectionFlags\(\)](#), [QwtPicker::setRubberBand\(\)](#), [QwtPicker::setTrackerMode\(\)](#)
[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#), [setZoomBase\(\)](#)

12.66.4 Member Function Documentation**12.66.4.1 bool QwtPlotZoomer::accept (QwtPolygon & *pa*) const**
[protected, virtual]

Check and correct a selected rectangle.

Reject rectangles with a height or width < 2, otherwise expand the selected rectangle to a minimum size of 11x11 and accept it.

Returns

true If rect is accepted, or has been changed to a accepted rectangle.

Reimplemented from [QwtPicker](#).

12.66.4.2 void QwtPlotPicker::append (const QPoint & *pos*) [protected, virtual, inherited]

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

Parameters

<i>pos</i>	Additional point
------------	------------------

See also

[isActive](#), [begin\(\)](#), [end\(\)](#), [move\(\)](#), [appended\(\)](#)

Note

The [appended\(const QPoint &\)](#), [appended\(const QDoublePoint &\)](#) signals are emitted.

Reimplemented from [QwtPicker](#).

12.66.4.3 void QwtPicker::appended (const QPoint & *pos*) [signal, inherited]

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

Parameters

<i>pos</i>	Position of the appended point.
------------	---------------------------------

See also

[append\(\). moved\(\)](#)

12.66.4.4 `void QwtPlotPicker::appended (const QwtDoublePoint & pos)`
[signal, inherited]

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

Parameters

<i>pos</i> Position of the appended point.
--

See also

[append\(\). moved\(\)](#)

12.66.4.5 `void QwtPlotZoomer::begin ()` **[protected, virtual]**

Rejects selections, when the stack depth is too deep, or the zoomed rectangle is [minZoomSize\(\)](#).

See also

[minZoomSize\(\)](#), [maxStackDepth\(\)](#)

Reimplemented from [QwtPicker](#).

12.66.4.6 `QwtPlotCanvas * QwtPlotPicker::canvas ()` **[inherited]**

Return observed plot canvas.

12.66.4.7 `const QwtPlotCanvas * QwtPlotPicker::canvas () const`
[inherited]

Return Observed plot canvas.

12.66.4.8 `void QwtPicker::changed (const QwtPolygon & pa)` **[signal, inherited]**

A signal emitted when the active selection has been changed. This might happen when the observed widget is resized.

Parameters

<i>pa</i>	Changed selection
-----------	-------------------

See also

[stretchSelection\(\)](#)

12.66.4.9 `void QwtPicker::drawRubberBand (QPainter * painter) const`
[virtual, inherited]

Draw a rubberband , depending on [rubberBand\(\)](#) and [selectionFlags\(\)](#)

Parameters

<i>painter</i>	Painter, initialized with clip rect
----------------	-------------------------------------

See also

[rubberBand\(\)](#), [RubberBand](#), [selectionFlags\(\)](#)

12.66.4.10 `void QwtPicker::drawTracker (QPainter * painter) const`
[virtual, inherited]

Draw the tracker

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[trackerRect\(\)](#), [trackerText\(\)](#)

12.66.4.11 `bool QwtPlotZoomer::end (bool ok = true)` **[protected, virtual]**

Expand the selected rectangle to [minZoomSize\(\)](#) and zoom in if accepted.

See also

[accept\(\)](#), [minZoomSize\(\)](#)

Reimplemented from [QwtPlotPicker](#).

12.66.4.12 `bool QwtPicker::eventFilter (QObject * o, QEvent * e)`
[virtual, inherited]

Event filter.

When `isEnabled()` == true all events of the observed widget are filtered. Mouse and keyboard events are translated into `widgetMouse-` and `widgetKey-` and `widgetWheel-` events. Paint and Resize events are handled to keep rubberband and tracker up to date.

See also

`event()`, `widgetMousePressEvent()`, `widgetMouseReleaseEvent()`, `widgetMouseDoubleClickEvent()`, `widgetMouseMoveEvent()`, `widgetWheelEvent()`, `widgetKeyPressEvent()`, `widgetKeyReleaseEvent()`

12.66.4.13 void QwtEventPattern::initKeyPattern () [inherited]

Set default mouse patterns.

See also

[KeyPatternCode](#)

12.66.4.14 void QwtEventPattern::initMousePattern (int numButtons) [inherited]

Set default mouse patterns, depending on the number of mouse buttons

Parameters

<i>numButtons</i>	Number of mouse buttons (<= 3)
-------------------	----------------------------------

See also

[MousePatternCode](#)

12.66.4.15 QwtDoubleRect QwtPlotPicker::invTransform (const QRect & rect) const [protected, inherited]

Translate a rectangle from pixel into plot coordinates

Returns

Rectangle in plot coordinates

See also

[QwtPlotPicker::transform\(\)](#)

12.66.4.16 QwtDoublePoint QwtPlotPicker::invTransform (const QPoint & *pos*) const [protected, inherited]

Translate a point from pixel into plot coordinates

Returns

Point in plot coordinates

See also

[QwtPlotPicker::transform\(\)](#)

12.66.4.17 bool QwtPicker::isActive () const [inherited]

A picker is active between [begin\(\)](#) and [end\(\)](#).

Returns

true if the selection is active.

12.66.4.18 bool QwtPicker::isEnabled () const [inherited]

Returns

true when enabled, false otherwise

See also

[setEnabled\(\)](#), [eventFilter\(\)](#)

12.66.4.19 bool QwtEventPattern::keyMatch (uint *pattern*, const QKeyEvent * *e*) const [inherited]

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.66.4.20 `bool QwtEventPattern::keyMatch (const KeyPattern & pattern,
const QKeyEvent * e) const` `[protected, virtual,
inherited]`

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

<i>pattern</i>	Key event pattern
<i>e</i>	Key event

Returns

true if matches

See also

[mouseMatch\(\)](#)

12.66.4.21 `const QwtArray< QwtEventPattern::KeyPattern > &
QwtEventPattern::keyPattern () const` `[inherited]`

Return key patterns.

12.66.4.22 `QwtArray< QwtEventPattern::KeyPattern > &
QwtEventPattern::keyPattern ()` `[inherited]`

Return Key patterns.

12.66.4.23 `int QwtPlotZoomer::maxStackDepth () const`

Returns

Maximal depth of the zoom stack.

See also

[setMaxStackDepth\(\)](#)

12.66.4.24 `QwtDoubleSize QwtPlotZoomer::minZoomSize () const`
[protected, virtual]

Limit zooming by a minimum rectangle.

Returns

[zoomBase\(\).width\(\)](#) / 10e4, [zoomBase\(\).height\(\)](#) / 10e4

12.66.4.25 `bool QwtEventPattern::mouseMatch (uint pattern, const`
`QMouseEvent * e) const` **[inherited]**

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Index of the event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.66.4.26 `bool QwtEventPattern::mouseMatch (const MousePattern`
`& pattern, const QMouseEvent * e) const` **[protected,**
virtual, inherited]

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(`Qt::KeyButtonMask`) are set.

Parameters

<i>pattern</i>	Mouse event pattern
<i>e</i>	Mouse event

Returns

true if matches

See also

[keyMatch\(\)](#)

12.66.4.27 `QwtArray< QwtEventPattern::MousePattern > &
QwtEventPattern::mousePattern () [inherited]`

Return ,ouse patterns.

12.66.4.28 `const QwtArray< QwtEventPattern::MousePattern > &
QwtEventPattern::mousePattern () const [inherited]`

Return mouse patterns.

12.66.4.29 `void QwtPlotPicker::move (const QPoint & pos) [protected,
virtual, inherited]`

Move the last point of the selection

Parameters

<i>pos</i>	New position
------------	--------------

See also

[isActive](#), [begin\(\)](#), [end\(\)](#), [append\(\)](#)

Note

The [moved\(const QPoint &\)](#), [moved\(const QDoublePoint &\)](#) signals are emitted.

Reimplemented from [QwtPicker](#).

12.66.4.30 `void QwtPlotZoomer::move (double x, double y) [virtual,
slot]`

Move the the current zoom rectangle.

Parameters

<i>x</i>	X value
<i>y</i>	Y value

See also

QwtDoubleRect::move()

Note

The changed rectangle is limited by the zoom base

12.66.4.31 void QwtPlotZoomer::moveBy (double *dx*, double *dy*) [slot]

Move the current zoom rectangle.

Parameters

<i>dx</i>	X offset
<i>dy</i>	Y offset

Note

The changed rectangle is limited by the zoom base

12.66.4.32 void QwtPicker::moved (const QPoint & *pos*) [signal, inherited]

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

Parameters

<i>pos</i>	Position of the moved last point of the selection.
------------	--

See also

[move\(\)](#), [appended\(\)](#)

12.66.4.33 void QwtPlotPicker::moved (const QwtDoublePoint & *pos*) [signal, inherited]

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

Parameters

<i>pos</i>	Position of the moved last point of the selection.
------------	--

See also

[move\(\)](#), [appended\(\)](#)

12.66.4.34 QWidget * QwtPicker::parentWidget () [inherited]

Return the parent widget, where the selection happens.

12.66.4.35 const QWidget * QwtPicker::parentWidget () const [inherited]

Return the parent widget, where the selection happens.

12.66.4.36 QRect QwtPicker::pickRect () const [virtual, inherited]

Find the area of the observed widget, where selection might happen.

Returns

QFrame::contentsRect() if it is a QFrame, QWidget::rect() otherwise.

12.66.4.37 QwtPlot * QwtPlotPicker::plot () [inherited]

Return plot widget, containing the observed plot canvas.

12.66.4.38 const QwtPlot * QwtPlotPicker::plot () const [inherited]

Return plot widget, containing the observed plot canvas.

12.66.4.39 void QwtPlotZoomer::rescale () [protected, virtual]

Adjust the observed plot to [zoomRect\(\)](#)

Note

Initiates [QwtPlot::replot](#)

12.66.4.40 `void QwtPicker::reset () [protected, virtual, inherited]`

Reset the state machine and terminate (end(false)) the selection

12.66.4.41 `QwtPicker::ResizeMode QwtPicker::resizeMode () const [inherited]`

Returns

Resize mode

See also

[setResizeMode\(\)](#), [ResizeMode](#)

12.66.4.42 `QwtPicker::RubberBand QwtPicker::rubberBand () const [inherited]`

Returns

Rubberband style

See also

[setRubberBand\(\)](#), [RubberBand](#), [rubberBandPen\(\)](#)

12.66.4.43 `QPen QwtPicker::rubberBandPen () const [inherited]`

Returns

Rubberband pen

See also

[setRubberBandPen\(\)](#), [rubberBand\(\)](#)

12.66.4.44 `const QWidget * QwtPicker::rubberBandWidget () const [protected, inherited]`

Returns

Widget displaying the rubberband

12.66.4.45 `QwtDoubleRect QwtPlotPicker::scaleRect () const`
[protected, inherited]

Return normalized bounding rect of the axes

See also

[QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#).

12.66.4.46 `void QwtPicker::selected (const QwtPolygon & pa)` **[signal, inherited]**

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

Parameters

<i>pa</i>	Selected points
-----------	-----------------

12.66.4.47 `void QwtPlotPicker::selected (const QwtArray< QwtDoublePoint > & pa)` **[signal, inherited]**

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

Parameters

<i>pa</i>	Selected points
-----------	-----------------

12.66.4.48 `void QwtPlotPicker::selected (const QwtDoublePoint & pos)`
[signal, inherited]

A signal emitted in case of [selectionFlags\(\)](#) & PointSelection.

Parameters

<i>pos</i>	Selected point
------------	----------------

12.66.4.49 `void QwtPlotPicker::selected (const QwtDoubleRect & rect)`
[signal, inherited]

A signal emitted in case of [selectionFlags\(\)](#) & RectSelection.

Parameters

<i>rect</i>	Selected rectangle
-------------	--------------------

12.66.4.50 `const QwtPolygon & QwtPicker::selection () const`
[inherited]

Return Selected points.

12.66.4.51 `int QwtPicker::selectionFlags () const` **[inherited]**

Returns

Selection flags, an Or'd value of SelectionType, RectSelectionType and SelectionMode.

See also

[setSelectionFlags\(\)](#), [SelectionType](#), [RectSelectionType](#), [SelectionMode](#)

12.66.4.52 `void QwtPlotZoomer::setAxis (int xAxis, int yAxis)` **[virtual]**

Reinitialize the axes, and set the zoom base to their scales.

Parameters

<i>xAxis</i>	X axis
<i>yAxis</i>	Y axis

Reimplemented from [QwtPlotPicker](#).

12.66.4.53 `void QwtPicker::setEnabled (bool enabled)` **[virtual, inherited]**

En/disable the picker.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

<i>enabled</i>	true or false
----------------	---------------

See also

[isEnabled\(\)](#), [eventFilter\(\)](#)

12.66.4.54 `void QwtEventPattern::setKeyPattern (const QwtArray< KeyPattern > & pattern) [inherited]`

Change the key event patterns.

12.66.4.55 `void QwtEventPattern::setKeyPattern (uint pattern, int key, int state = Qt::NoButton) [inherited]`

Change one key pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>key</i>	Key
<i>state</i>	State

See also

[QKeyEvent](#)

12.66.4.56 `void QwtPlotZoomer::setMaxStackDepth (int depth)`

Limit the number of recursive zoom operations to *depth*.

A value of -1 set the depth to unlimited, 0 disables zooming. If the current zoom rectangle is below *depth*, the plot is unzoomed.

Parameters

<i>depth</i>	Maximum for the stack depth
--------------	-----------------------------

See also

[maxStackDepth\(\)](#)

Note

depth doesn't include the zoom base, so [zoomStack\(\).count\(\)](#) might be [maxStackDepth\(\)](#) + 1.

12.66.4.57 `void QwtEventPattern::setMousePattern (const QwtArray< MousePattern > & pattern) [inherited]`

Change the mouse event patterns.

12.66.4.58 void QwtEventPattern::setMousePattern (uint *pattern*, int *button*, int *state* = Qt::NoButton) [inherited]

Change one mouse pattern

Parameters

<i>pattern</i>	Index of the pattern
<i>button</i>	Button
<i>state</i>	State

See also

QMouseEvent

12.66.4.59 void QwtPicker::setResizeMode (ResizeMode *mode*)
[virtual, inherited]

Set the resize mode.

The resize mode controls what to do with the selected points of an active selection when the observed widget is resized.

Stretch means the points are scaled according to the new size, KeepSize means the points remain unchanged.

The default mode is Stretch.

Parameters

<i>mode</i>	Resize mode
-------------	-------------

See also

[resizeMode\(\)](#), [ResizeMode](#)

12.66.4.60 void QwtPicker::setRubberBand (RubberBand *rubberBand*)
[virtual, inherited]

Set the rubberband style

Parameters

<i>rubberBand</i>	Rubberband style The default value is NoRubberBand.
-------------------	---

See also

[rubberBand\(\)](#), [RubberBand](#), [setRubberBandPen\(\)](#)

12.66.4.61 void QwtPicker::setRubberBandPen (const QPen & *pen*)
[virtual, inherited]

Set the pen for the rubberband

Parameters

<i>pen</i>	Rubberband pen
------------	----------------

See also

[rubberBandPen\(\)](#), [setRubberBand\(\)](#)

12.66.4.62 void QwtPlotZoomer::setSelectionFlags (int *flags*) [virtual]

Set the selection flags

Parameters

<i>flags</i>	Or'd value of QwtPicker::RectSelectionType and QwtPicker::SelectionMode . The default value is QwtPicker::RectSelection & QwtPicker::ClickSelection.
--------------	--

See also

[selectionFlags\(\)](#), [SelectionMode](#), [RectSelectionType](#), [SelectionMode](#)

Note

QwtPicker::RectSelection will be auto added.

Reimplemented from [QwtPicker](#).

12.66.4.63 void QwtPicker::setTrackerFont (const QFont & *font*)
[virtual, inherited]

Set the font for the tracker

Parameters

<i>font</i>	Tracker font
-------------	--------------

See also

[trackerFont\(\)](#), [setTrackerMode\(\)](#), [setTrackerPen\(\)](#)

12.66.4.64 void QwtPicker::setTrackerMode (DisplayMode *mode*) [virtual, inherited]

Set the display mode of the tracker.

A tracker displays information about current position of the cursor as a string. The display mode controls if the tracker has to be displayed whenever the observed widget has focus and cursor (AlwaysOn), never (AlwaysOff), or only when the selection is active (ActiveOnly).

Parameters

<i>mode</i>	Tracker display mode
-------------	----------------------

Warning

In case of AlwaysOn, mouseTracking will be enabled for the observed widget.

See also

[trackerMode\(\)](#), [DisplayMode](#)

12.66.4.65 void QwtPicker::setTrackerPen (const QPen &*pen*) [virtual, inherited]

Set the pen for the tracker

Parameters

<i>pen</i>	Tracker pen
------------	-------------

See also

[trackerPen\(\)](#), [setTrackerMode\(\)](#), [setTrackerFont\(\)](#)

12.66.4.66 void QwtPlotZoomer::setZoomBase (bool *doReplot* = *true*) [virtual]

Reinitialized the zoom stack with [scaleRect\(\)](#) as base.

Parameters

<i>doReplot</i>	Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.
-----------------	--

See also

[zoomBase\(\)](#), [scaleRect\(\)](#) [QwtPlot::autoReplot\(\)](#), [QwtPlot::replot\(\)](#).

12.66.4.67 `void QwtPlotZoomer::setZoomBase (const QwtDoubleRect & base) [virtual]`

Set the initial size of the zoomer.

base is united with the current [scaleRect\(\)](#) and the zoom stack is reinitialized with it as zoom base. plot is zoomed to [scaleRect\(\)](#).

Parameters

<i>base</i>	Zoom base
-------------	-----------

See also

[zoomBase\(\)](#), [scaleRect\(\)](#)

12.66.4.68 `void QwtPlotZoomer::setZoomStack (const QStack< QwtDoubleRect > &, int zoomRectIndex = -1)`

Assign a zoom stack.

In combination with other types of navigation it might be useful to modify to manipulate the complete zoom stack.

Parameters

<i>zoomStack</i>	New zoom stack
<i>zoomRectIndex</i>	Index of the current position of zoom stack. In case of -1 the current position is at the top of the stack.

Note

The zoomed signal might be emitted.

See also

[zoomStack\(\)](#), [zoomRectIndex\(\)](#)

12.66.4.69 `QwtPickerMachine * QwtPicker::stateMachine (int flags) const [protected, virtual, inherited]`

Create a state machine depending on the selection flags.

- PointSelection | ClickSelection
QwtPickerClickPointMachine()
- PointSelection | DragSelection
QwtPickerDragPointMachine()

- RectSelection | ClickSelection
QwtPickerClickRectMachine()
- RectSelection | DragSelection
QwtPickerDragRectMachine()
- PolygonSelection
QwtPickerPolygonMachine()

See also

[setSelectionFlags\(\)](#)

12.66.4.70 void QwtPicker::stretchSelection (const QSize & *oldSize*, const QSize & *newSize*) [protected, virtual, inherited]

Scale the selection by the ratios of *oldSize* and *newSize* The [changed\(\)](#) signal is emitted.

Parameters

<i>oldSize</i>	Previous size
<i>newSize</i>	Current size

See also

[ResizeMode](#), [setResizeMode\(\)](#), [resizeMode\(\)](#)

12.66.4.71 QFont QwtPicker::trackerFont () const [inherited]

Returns

Tracker font

See also

[setTrackerFont\(\)](#), [trackerMode\(\)](#), [trackerPen\(\)](#)

12.66.4.72 QwtPicker::DisplayMode QwtPicker::trackerMode () const [inherited]

Returns

Tracker display mode

See also

[setTrackerMode\(\)](#), [DisplayMode](#)

12.66.4.73 QPen QwtPicker::trackerPen () const [inherited]

Returns

Tracker pen

See also

[setTrackerPen\(\)](#), [trackerMode\(\)](#), [trackerFont\(\)](#)

12.66.4.74 QPoint QwtPicker::trackerPosition () const [inherited]

Returns

Current position of the tracker

12.66.4.75 QRect QwtPicker::trackerRect (const QFont & *font*) const [inherited]

Calculate the bounding rectangle for the tracker text from the current position of the tracker

Parameters

<i>font</i>	Font of the tracker text
-------------	--------------------------

Returns

Bounding rectangle of the tracker text

See also

[trackerPosition\(\)](#)

12.66.4.76 QString QwtPlotPicker::trackerText (const QwtDoublePoint & *pos*) const [protected, virtual, inherited]

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

<i>pos</i>	Position
------------	----------

Returns

Position string

12.66.4.77 `QwtText QwtPlotPicker::trackerText (const QPoint & pos) const`
[protected, virtual, inherited]

Translate a pixel position into a position string

Parameters

<i>pos</i>	Position in pixel coordinates
------------	-------------------------------

Returns

Position string

Reimplemented from [QwtPicker](#).

12.66.4.78 `const QWidget * QwtPicker::trackerWidget () const`
[protected, inherited]

Returns

Widget displaying the tracker text

12.66.4.79 `QPoint QwtPlotPicker::transform (const QwtDoublePoint & pos)`
const [protected, inherited]

Translate a point from plot into pixel coordinates

Returns

Point in pixel coordinates

See also

[QwtPlotPicker::invTransform\(\)](#)

12.66.4.80 `QRect QwtPlotPicker::transform (const QwtDoubleRect & rect)
const [protected, inherited]`

Translate a rectangle from plot into pixel coordinates

Returns

Rectangle in pixel coordinates

See also

[QwtPlotPicker::invTransform\(\)](#)

12.66.4.81 `void QwtPicker::transition (const QEvent * e) [protected,
virtual, inherited]`

Passes an event to the state machine and executes the resulting commands. Append and Move commands use the current position of the cursor (`QCursor::pos()`).

Parameters

<i>e</i>	Event
----------	-------

12.66.4.82 `void QwtPicker::updateDisplay () [protected, virtual,
inherited]`

Update the state of rubberband and tracker label.

12.66.4.83 `void QwtPlotZoomer::widgetKeyPressEvent (QKeyEvent * ke)
[protected, virtual]`

Qt::Key_Plus zooms in, Qt::Key_Minus zooms out one position on the zoom stack, Qt::Key_Escape zooms out to the zoom base.

Changes the current position on the stack, but doesn't pop any rectangle.

Note

The keys codes can be changed, using [QwtEventPattern::setKeyPattern](#): 3, 4, 5

Reimplemented from [QwtPicker](#).

12.66.4.84 `void QwtPicker::widgetKeyReleaseEvent (QKeyEvent * ke)
[protected, virtual, inherited]`

Handle a key release event for the observed widget.

Passes the event to the state machine.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [stateMachine\(\)](#)

12.66.4.85 void QwtPicker::widgetLeaveEvent (QEvent *) [protected, virtual, inherited]

Handle a leave event for the observed widget.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.66.4.86 void QwtPicker::widgetMouseDoubleClickEvent (QMouseEvent * me) [protected, virtual, inherited]

Handle mouse double click event for the observed widget.

Empty implementation, does nothing.

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.66.4.87 void QwtPicker::widgetMouseMoveEvent (QMouseEvent * e) [protected, virtual, inherited]

Handle a mouse move event for the observed widget.

Move the last point of the selection in case of [isActive\(\)](#) == true

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMousePressEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.66.4.88 void QwtPicker::widgetMouseEvent (QMouseEvent * *e*)
[protected, virtual, inherited]

Handle a mouse press event for the observed widget.

Begin and/or end a selection depending on the selection flags.

See also

[QwtPicker](#), [selectionFlags\(\)](#)
[eventFilter\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetWheelEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.66.4.89 void QwtPlotZoomer::widgetMouseReleaseEvent (QMouseEvent * *me*) [protected, virtual]

Qt::MidButton zooms out one position on the zoom stack, Qt::RightButton to the zoom base.

Changes the current position on the stack, but doesn't pop any rectangle.

Note

The mouse events can be changed, using [QwtEventPattern::setMousePattern](#): 2, 1

Reimplemented from [QwtPicker](#).

12.66.4.90 void QwtPicker::widgetWheelEvent (QWheelEvent * *e*)
[protected, virtual, inherited]

Handle a wheel event for the observed widget.

Move the last point of the selection in case of [isActive\(\)](#) == true

See also

[eventFilter\(\)](#), [widgetMouseEvent\(\)](#), [widgetMouseReleaseEvent\(\)](#), [widgetMouseDoubleClickEvent\(\)](#), [widgetMouseMoveEvent\(\)](#), [widgetKeyPressEvent\(\)](#), [widgetKeyReleaseEvent\(\)](#)

12.66.4.91 int QwtPlotPicker::xAxis () const [inherited]

Return x axis.

12.66.4.92 `int QwtPlotPicker::yAxis () const` `[inherited]`

Return y axis.

12.66.4.93 `void QwtPlotZoomer::zoom (int offset)` `[virtual, slot]`

Zoom in or out.

Activate a rectangle on the zoom stack with an offset relative to the current position. Negative values of offset will zoom out, positive zoom in. A value of 0 zooms out to the zoom base.

Parameters

<i>offset</i>	Offset relative to the current position of the zoom stack.
---------------	--

Note

The zoomed signal is emitted.

See also

[zoomRectIndex\(\)](#)

12.66.4.94 `void QwtPlotZoomer::zoom (const QwtDoubleRect & rect)`
`[virtual, slot]`

Zoom in.

Clears all rectangles above the current position of the zoom stack and pushes the intersection of [zoomRect\(\)](#) and the normalized rect on it.

Note

If the maximal stack depth is reached, zoom is ignored.
The zoomed signal is emitted.

12.66.4.95 `QwtDoubleRect QwtPlotZoomer::zoomBase () const`**Returns**

Initial rectangle of the zoomer

See also

[setZoomBase\(\)](#), [zoomRect\(\)](#)

12.66.4.96 `void QwtPlotZoomer::zoomed (const QwtDoubleRect & rect)`
[**signal**]

A signal emitting the [zoomRect\(\)](#), when the plot has been zoomed in or out.

Parameters

<i>rect</i> Current zoom rectangle.

12.66.4.97 `QwtDoubleRect QwtPlotZoomer::zoomRect () const`

Rectangle at the current position on the zoom stack.

See also

[zoomRectIndex\(\)](#), [scaleRect\(\)](#).

12.66.4.98 `uint QwtPlotZoomer::zoomRectIndex () const`

Returns

Index of current position of zoom stack.

12.66.4.99 `const QwtZoomStack & QwtPlotZoomer::zoomStack () const`

Return the zoom stack. [zoomStack\(\)\[0\]](#) is the zoom base, [zoomStack\(\)\[1\]](#) the first zoomed rectangle.

See also

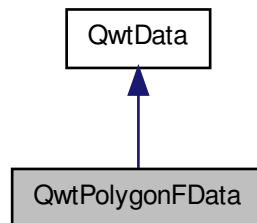
[setZoomStack\(\)](#), [zoomRectIndex\(\)](#)

12.67 QwtPolygonFData Class Reference

Data class containing a single `QwtArray<QwtDoublePoint>` object.

```
#include <qwt_data.h>
```

Inheritance diagram for QwtPolygonFData:



Public Member Functions

- virtual QwtDoubleRect [boundingRect](#) () const
- virtual [QwtData](#) * [copy](#) () const
- const QPolygonF & [data](#) () const
- [QwtPolygonFData](#) & [operator=](#) (const [QwtPolygonFData](#) &)
- [QwtPolygonFData](#) (const QPolygonF &)
- virtual size_t [size](#) () const
- virtual double [x](#) (size_t i) const
- virtual double [y](#) (size_t i) const

12.67.1 Detailed Description

Data class containing a single QwtArray<QwtDoublePoint> object.

12.67.2 Constructor & Destructor Documentation

12.67.2.1 QwtPolygonFData::QwtPolygonFData (const QPolygonF & *polygon*)

Constructor

Parameters

<i>polygon</i>	Polygon data
----------------	--------------

See also

[QwtPlotCurve::setData\(\)](#)

12.67.3 Member Function Documentation

12.67.3.1 QwtDoubleRect QwtData::boundingRect () const [virtual, inherited]

Returns the bounding rectangle of the data. If there is no bounding rect, like for empty data the rectangle is invalid: QwtDoubleRect::isValid() == false

Warning

This is an slow implementation iterating over all points. It is intended to be overloaded by derived classes. In case of auto scaling [boundingRect\(\)](#) is called for every replot, so it might be worth to implement a cache, or use $x(0)$, $x(\text{size}() - 1)$ for ordered data ...

Reimplemented in [QwtArrayData](#), and [QwtCPointerData](#).

12.67.3.2 QwtData * QwtPolygonFData::copy () const [virtual]

Returns

Pointer to a copy (virtual copy constructor)

Implements [QwtData](#).

12.67.3.3 const QPolygonF & QwtPolygonFData::data () const

Returns

Point array

12.67.3.4 QwtPolygonFData & QwtPolygonFData::operator= (const QwtPolygonFData & *data*)

Assignment.

12.67.3.5 size_t QwtPolygonFData::size () const [virtual]

Returns

Size of the data set

Implements [QwtData](#).

12.67.3.6 double QwtPolygonFData::x (size_t *i*) const [virtual]Return the x value of data point *i***Parameters**

<i>i</i>	Index
----------	-------

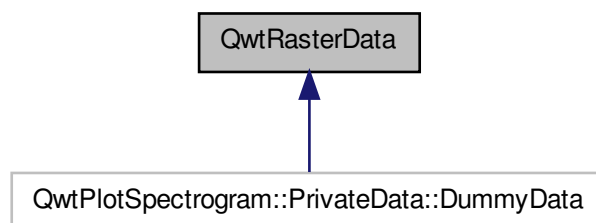
Returnsx X value of data point *i*Implements [QwtData](#).**12.67.3.7 double QwtPolygonFData::y (size_t *i*) const [virtual]**Return the y value of data point *i***Parameters**

<i>i</i>	Index
----------	-------

Returnsy Y value of data point *i*Implements [QwtData](#).**12.68 QwtRasterData Class Reference**[QwtRasterData](#) defines an interface to any type of raster data.

#include <qwt_raster_data.h>

Inheritance diagram for QwtRasterData:



Public Types

- enum [ConrecAttribute](#) {
 IgnoreAllVerticesOnLevel = 1,
 IgnoreOutOfRange = 2 }
- typedef QMap< double, QPolygonF > **ContourLines**

Public Member Functions

- QwtDoubleRect [boundingRect](#) () const
- virtual ContourLines [contourLines](#) (const QwtDoubleRect &rect, const QSize &raster, const QList< double > &levels, int flags) const
- virtual [QwtRasterData](#) * [copy](#) () const =0
- virtual void [discardRaster](#) ()
- virtual void [initRaster](#) (const QwtDoubleRect &, const QSize &raster)
- [QwtRasterData](#) ()
- [QwtRasterData](#) (const QwtDoubleRect &)
- virtual [QwtDoubleInterval](#) [range](#) () const =0
- virtual QSize [rasterHint](#) (const QwtDoubleRect &) const
- virtual void [setBoundingRect](#) (const QwtDoubleRect &)
- virtual double [value](#) (double x, double y) const =0
- virtual [~QwtRasterData](#) ()

12.68.1 Detailed Description

[QwtRasterData](#) defines an interface to any type of raster data. [QwtRasterData](#) is an abstract interface, that is used by [QwtPlotRasterItem](#) to find the values at the pixels of its raster.

Often a raster item is used to display values from a matrix. Then the derived raster data class needs to implement some sort of resampling, that maps the raster of the matrix into the requested raster of the raster item (depending on resolution and scales of the canvas).

12.68.2 Member Enumeration Documentation**12.68.2.1 enum QwtRasterData::ConrecAttribute**

Attribute to modify the contour algorithm.

12.68.3 Constructor & Destructor Documentation**12.68.3.1 QwtRasterData::QwtRasterData ()**

Constructor.

12.68.3.2 QwtRasterData::QwtRasterData (const QwtDoubleRect & *boundingRect*)

Constructor

Parameters

<i>boundingRect</i>	Bounding rectangle
---------------------	--------------------

See also

[setBoundingRect\(\)](#)

12.68.3.3 QwtRasterData::~QwtRasterData () [virtual]

Destructor.

12.68.4 Member Function Documentation

12.68.4.1 QwtDoubleRect QwtRasterData::boundingRect () const

Returns

Bounding rectangle

See also

[boundingRect\(\)](#)

12.68.4.2 QwtRasterData::ContourLines QwtRasterData::contourLines (const QwtDoubleRect & *rect*, const QSize & *raster*, const QList< double > & *levels*, int *flags*) const [virtual]

Calculate contour lines

An adaption of CONREC, a simple contouring algorithm. <http://local.wasp.uwa.edu.au/~pbourke/pa>

12.68.4.3 virtual QwtRasterData* QwtRasterData::copy () const [pure virtual]

Clone the data.

12.68.4.4 void QwtRasterData::discardRaster () [virtual]

Discard a raster.

After the composition of an image [QwtPlotSpectrogram](#) calls [discardRaster\(\)](#).

The default implementation does nothing, but if data has been loaded in [initRaster\(\)](#), it could be deleted now.

See also

[initRaster\(\)](#), [value\(\)](#)

12.68.4.5 void QwtRasterData::initRaster (const QwtDoubleRect &, const QSize & raster) [virtual]

Initialize a raster.

Before the composition of an image [QwtPlotSpectrogram](#) calls [initRaster](#), announcing the area and its resolution that will be requested.

The default implementation does nothing, but for data sets that are stored in files, it might be a good idea to reimplement [initRaster](#), where the data is resampled and loaded into memory.

Parameters

<i>rect</i>	Area of the raster
<i>raster</i>	Number of horizontal and vertical pixels

See also

[initRaster\(\)](#), [value\(\)](#)

12.68.4.6 virtual QwtDoubleInterval QwtRasterData::range () const [pure virtual]

Returns

the range of the values

12.68.4.7 QSize QwtRasterData::rasterHint (const QwtDoubleRect &) const [virtual]

Find the raster of the data for an area.

The resolution is the number of horizontal and vertical pixels that the data can return for an area. An invalid resolution indicates that the data can return values for any detail level.

The resolution will limit the size of the image that is rendered from the data. F.e. this might be important when printing a spectrogram to a A0 printer with 600 dpi.

The default implementation returns an invalid resolution (size)

Parameters

<i>rect</i>	In most implementations the resolution of the data doesn't depend on the requested rectangle.
-------------	---

Returns

Resolution, as number of horizontal and vertical pixels

12.68.4.8 void QwtRasterData::setBoundingRect (const QwtDoubleRect & boundingRect) [virtual]

Set the bounding rect (== area, un plot coordinates)

Parameters

<i>boundingRect</i>	Bounding rectangle
---------------------	--------------------

See also

[boundingRect\(\)](#)

12.68.4.9 virtual double QwtRasterData::value (double x, double y) const [pure virtual]

Returns

the value at a raster position

Parameters

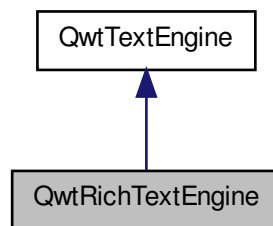
<i>x</i>	X value in plot coordinates
<i>y</i>	Y value in plot coordinates

12.69 QwtRichTextEngine Class Reference

A text engine for Qt rich texts.

```
#include <qwt_text_engine.h>
```

Inheritance diagram for QwtRichTextEngine:



Public Member Functions

- virtual void [draw](#) (QPainter *painter, const QRect &rect, int flags, const QString &text) const
- virtual int [heightForWidth](#) (const QFont &font, int flags, const QString &text, int width) const
- virtual bool [mightRender](#) (const QString &) const
- [QwtRichTextEngine](#) ()
- virtual void [textMargins](#) (const QFont &, const QString &, int &left, int &right, int &top, int &bottom) const
- virtual QSize [textSize](#) (const QFont &font, int flags, const QString &text) const

12.69.1 Detailed Description

A text engine for Qt rich texts. [QwtRichTextEngine](#) renders Qt rich texts using the classes of the Scribe framework of Qt.

12.69.2 Constructor & Destructor Documentation

12.69.2.1 QwtRichTextEngine::QwtRichTextEngine ()

Constructor.

12.69.3 Member Function Documentation

12.69.3.1 `void QwtRichTextEngine::draw (QPainter * painter, const QRect & rect, int flags, const QString & text) const` **[virtual]**

Draw the text in a clipping rectangle

Parameters

<i>painter</i>	Painter
<i>rect</i>	Clipping rectangle
<i>flags</i>	Bitwise OR of the flags like in for QPainter::drawText
<i>text</i>	Text to be rendered

Implements [QwtTextEngine](#).

12.69.3.2 `int QwtRichTextEngine::heightForWidth (const QFont & font, int flags, const QString & text, int width) const` **[virtual]**

Find the height for a given width

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered
<i>width</i>	Width

Returns

Calculated height

Implements [QwtTextEngine](#).

12.69.3.3 `bool QwtRichTextEngine::mightRender (const QString & text) const` **[virtual]**

Test if a string can be rendered by this text engine

Parameters

<i>text</i>	Text to be tested
-------------	-------------------

Returns

QStyleSheet::mightBeRichText(text);

Implements [QwtTextEngine](#).

12.69.3.4 void QwtRichTextEngine::textMargins (const QFont &, const QString &, int & *left*, int & *right*, int & *top*, int & *bottom*) const [virtual]

Return margins around the texts

Parameters

<i>left</i>	Return 0
<i>right</i>	Return 0
<i>top</i>	Return 0
<i>bottom</i>	Return 0

Implements [QwtTextEngine](#).

12.69.3.5 QSize QwtRichTextEngine::textSize (const QFont & *font*, int *flags*, const QString & *text*) const [virtual]

Returns the size, that is needed to render text

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered

Returns

Caluclated size

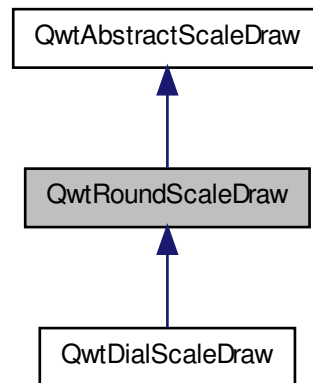
Implements [QwtTextEngine](#).

12.70 QwtRoundScaleDraw Class Reference

A class for drawing round scales.

```
#include <qwt_round_scale_draw.h>
```

Inheritance diagram for QwtRoundScaleDraw:



Public Types

- enum [ScaleComponent](#) {
 Backbone = 1,
 Ticks = 2,
 Labels = 4 }

Public Member Functions

- [QPoint](#) [center](#) () const
- virtual void [draw](#) (QPainter *, const [QPalette](#) &) const
- void [enableComponent](#) ([ScaleComponent](#), bool enable=true)
- virtual int [extent](#) (const [QPen](#) &, const [QFont](#) &) const
- bool [hasComponent](#) ([ScaleComponent](#)) const
- virtual [QwtText](#) [label](#) (double) const
- int [majTickLength](#) () const
- const [QwtScaleMap](#) & [map](#) () const
- int [minimumExtent](#) () const
- void [moveCenter](#) (const [QPoint](#) &)
- void [moveCenter](#) (int x, int y)
- [QwtRoundScaleDraw](#) & [operator=](#) (const [QwtRoundScaleDraw](#) &other)
- [QwtRoundScaleDraw](#) (const [QwtRoundScaleDraw](#) &)
- [QwtRoundScaleDraw](#) ()
- int [radius](#) () const

- const [QwtScaleDiv](#) & [scaleDiv](#) () const
- [QwtScaleMap](#) & [scaleMap](#) ()
- void [setAngleRange](#) (double angle1, double angle2)
- void [setMinimumExtent](#) (int)
- void [setRadius](#) (int radius)
- void [setScaleDiv](#) (const [QwtScaleDiv](#) &s)
- void [setSpacing](#) (int margin)
- void [setTickLength](#) ([QwtScaleDiv::TickType](#), int length)
- void [setTransformation](#) ([QwtScaleTransformation](#) *)
- int [spacing](#) () const
- int [tickLength](#) ([QwtScaleDiv::TickType](#)) const
- virtual [~QwtRoundScaleDraw](#) ()

Protected Member Functions

- virtual void [drawBackbone](#) (QPainter *p) const
- virtual void [drawLabel](#) (QPainter *p, double val) const
- virtual void [drawTick](#) (QPainter *p, double val, int len) const
- void [invalidateCache](#) ()
- const [QwtText](#) & [tickLabel](#) (const QFont &, double value) const

12.70.1 Detailed Description

A class for drawing round scales. [QwtRoundScaleDraw](#) can be used to draw round scales. The circle segment can be adjusted by [QwtRoundScaleDraw::setAngleRange\(\)](#). The geometry of the scale can be specified with [QwtRoundScaleDraw::moveCenter\(\)](#) and [QwtRoundScaleDraw::setRadius\(\)](#).

After a scale division has been specified as a [QwtScaleDiv](#) object using [QwtAbstractScaleDraw::setScaleDiv\(const QwtScaleDiv &s\)](#), the scale can be drawn with the [QwtAbstractScaleDraw::draw\(\)](#) member.

12.70.2 Member Enumeration Documentation

12.70.2.1 enum QwtAbstractScaleDraw::ScaleComponent [inherited]

Components of a scale

- Backbone
- Ticks
- Labels

See also

[enableComponent\(\)](#), [hasComponent](#)

12.70.3 Constructor & Destructor Documentation

12.70.3.1 QwtRoundScaleDraw::QwtRoundScaleDraw ()

Constructor.

The range of the scale is initialized to [0, 100], The center is set to (50, 50) with a radius of 50. The angle range is set to [-135, 135].

12.70.3.2 QwtRoundScaleDraw::QwtRoundScaleDraw (const QwtRoundScaleDraw & *other*)

Copy constructor.

12.70.3.3 QwtRoundScaleDraw::~QwtRoundScaleDraw () [virtual]

Destructor.

12.70.4 Member Function Documentation

12.70.4.1 QPoint QwtRoundScaleDraw::center () const

Get the center of the scale.

12.70.4.2 void QwtAbstractScaleDraw::draw (QPainter * *painter*, const QPalette & *palette*) const [virtual, inherited]

Draw the scale.

Parameters

<i>painter</i>	The painter
<i>palette</i>	Palette, text color is used for the labels, foreground color for ticks and backbone

12.70.4.3 void QwtRoundScaleDraw::drawBackbone (QPainter * *painter*) const [protected, virtual]

Draws the baseline of the scale

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[drawTick\(\)](#), [drawLabel\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.70.4.4 void QwtRoundScaleDraw::drawLabel (QPainter * *painter*, double *value*) const [**protected**, **virtual**]

Draws the label for a major scale tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value

See also

[drawTick\(\)](#), [drawBackbone\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.70.4.5 void QwtRoundScaleDraw::drawTick (QPainter * *painter*, double *value*, int *len*) const [**protected**, **virtual**]

Draw a tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value of the tick
<i>len</i>	Lenght of the tick

See also

[drawBackbone\(\)](#), [drawLabel\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.70.4.6 void QwtAbstractScaleDraw::enableComponent (ScaleComponent *component*, bool *enable* = *true*) [**inherited**]

En/Disable a component of the scale

Parameters

<i>component</i>	Scale component
<i>enable</i>	On/Off

See also

[hasComponent\(\)](#)

12.70.4.7 `int QwtRoundScaleDraw::extent (const QPen & pen, const QFont & font) const` **[virtual]**

Calculate the extent of the scale

The extent is the distance between the baseline to the outermost pixel of the scale draw. [radius\(\)](#) + [extent\(\)](#) is an upper limit for the radius of the bounding circle.

Parameters

<i>pen</i>	Pen that is used for painting backbone and ticks
<i>font</i>	Font used for painting the labels

See also

[setMinimumExtent\(\)](#), [minimumExtent\(\)](#)

Warning

The implemented algo is not too smart and calculates only an upper limit, that might be a few pixels too large

Implements [QwtAbstractScaleDraw](#).

12.70.4.8 `bool QwtAbstractScaleDraw::hasComponent (ScaleComponent component) const` **[inherited]**

Check if a component is enabled

See also

[enableComponent\(\)](#)

12.70.4.9 `void QwtAbstractScaleDraw::invalidateCache ()` **[protected, inherited]**

Invalidate the cache used by [QwtAbstractScaleDraw::tickLabel](#)

The cache is invalidated, when a new [QwtScaleDiv](#) is set. If the labels need to be changed. while the same [QwtScaleDiv](#) is set, [QwtAbstractScaleDraw::invalidateCache](#) needs to be called manually.

12.70.4.10 `QwtText QwtAbstractScaleDraw::label (double value) const` **[virtual, inherited]**

Convert a value into its representing label.

The value is converted to a plain text using `QLocale::system().toString(value)`. This method is often overloaded by applications to have individual labels.

Parameters

<i>value</i>	Value
--------------	-------

Returns

Label string.

Reimplemented in [QwtDialScaleDraw](#).

12.70.4.11 `int QwtAbstractScaleDraw::majTickLength () const` [*inherited*]

The same as `QwtAbstractScaleDraw::tickLength(QwtScaleDiv::MajorTick)`.

12.70.4.12 `const QwtScaleMap & QwtAbstractScaleDraw::map () const` [*inherited*]

Returns

Map how to translate between scale and pixel values

12.70.4.13 `int QwtAbstractScaleDraw::minimumExtent () const` [*inherited*]

Get the minimum extent

See also

[extent\(\)](#), [setMinimumExtent\(\)](#)

12.70.4.14 `void QwtRoundScaleDraw::moveCenter (const QPoint & center)`

Move the center of the scale draw, leaving the radius unchanged

Parameters

<i>center</i>	New center
---------------	------------

See also

[setRadius\(\)](#)

12.70.4.15 void QwtRoundScaleDraw::moveCenter (int x, int y) [inline]

Move the center of the scale draw, leaving the radius unchanged.

12.70.4.16 QwtRoundScaleDraw & QwtRoundScaleDraw::operator= (const QwtRoundScaleDraw & other)

Assignment operator.

12.70.4.17 int QwtRoundScaleDraw::radius () const

Get the radius

Radius is the radius of the backbone without ticks and labels.

See also

[setRadius\(\)](#), [extent\(\)](#)

12.70.4.18 const QwtScaleDiv & QwtAbstractScaleDraw::scaleDiv () const [inherited]

Returns

scale division

12.70.4.19 QwtScaleMap & QwtAbstractScaleDraw::scaleMap () [inherited]

Returns

Map how to translate between scale and pixel values

12.70.4.20 void QwtRoundScaleDraw::setAngleRange (double angle1, double angle2)

Adjust the baseline circle segment for round scales.

The baseline will be drawn from min(angle1,angle2) to max(angle1, angle2). The default setting is [-135, 135]. An angle of 0 degrees corresponds to the 12 o'clock position, and positive angles count in a clockwise direction.

Parameters

<i>angle1</i>	
<i>angle2</i>	boundaries of the angle interval in degrees.

Warning

- The angle range is limited to [-360, 360] degrees. Angles exceeding this range will be clipped.
- For angles more than 359 degrees above or below min(angle1, angle2), scale marks will not be drawn.
- If you need a counterclockwise scale, use QwtScaleDiv::setRange

12.70.4.21 void QwtAbstractScaleDraw::setMinimumExtent (int *minExtent*) [inherited]

Set a minimum for the extent.

The extent is calculated from the components of the scale draw. In situations, where the labels are changing and the layout depends on the extent (f.e scrolling a scale), setting an upper limit as minimum extent will avoid jumps of the layout.

Parameters

<i>minExtent</i>	Minimum extent
------------------	----------------

See also

[extent\(\)](#), [minimumExtent\(\)](#)

12.70.4.22 void QwtRoundScaleDraw::setRadius (int *radius*)

Change of radius the scale

Radius is the radius of the backbone without ticks and labels.

Parameters

<i>radius</i>	New Radius
---------------	------------

See also

[moveCenter\(\)](#)

12.70.4.23 void QwtAbstractScaleDraw::setScaleDiv (const QwtScaleDiv & *sd*) [inherited]

Change the scale division

Parameters

<i>sd</i>	New scale division
-----------	--------------------

12.70.4.24 void QwtAbstractScaleDraw::setSpacing (int *spacing*) [inherited]

Set the spacing between tick and labels.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

[spacing\(\)](#)

12.70.4.25 void QwtAbstractScaleDraw::setTickLength (QwtScaleDiv::TickType *tickType*, int *length*) [inherited]

Set the length of the ticks

Parameters

<i>tickType</i>	Tick type
<i>length</i>	New length

Warning

the length is limited to [0..1000]

12.70.4.26 void QwtAbstractScaleDraw::setTransformation (QwtScaleTransformation * *transformation*) [inherited]

Change the transformation of the scale

Parameters

<i>transformation</i>	New scale transformation
-----------------------	--------------------------

12.70.4.27 int QwtAbstractScaleDraw::spacing () const [inherited]

Get the spacing.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

See also

[setSpacing\(\)](#)

12.70.4.28 const QwtText & QwtAbstractScaleDraw::tickLabel (const QFont & font, double value) const [protected, inherited]

Convert a value into its representing label and cache it.

The conversion between value and label is called very often in the layout and painting code. Unfortunately the calculation of the label sizes might be slow (really slow for rich text in Qt4), so it's necessary to cache the labels.

Parameters

<i>font</i>	Font
<i>value</i>	Value

Returns

Tick label

12.70.4.29 int QwtAbstractScaleDraw::tickLength (QwtScaleDiv::TickType tickType) const [inherited]

Return the length of the ticks

See also

[setTickLength\(\)](#), [majTickLength\(\)](#)

12.71 QwtScaleArithmetic Class Reference

Arithmetic including a tolerance.

```
#include <qwt_scale_engine.h>
```

Static Public Member Functions

- static double [ceil125](#) (double x)

- static double [ceilEps](#) (double value, double intervalSize)
- static int [compareEps](#) (double value1, double value2, double intervalSize)
- static double [divideEps](#) (double interval, double steps)
- static double [floor125](#) (double x)
- static double [floorEps](#) (double value, double intervalSize)

12.71.1 Detailed Description

Arithmetic including a tolerance.

12.71.2 Member Function Documentation

12.71.2.1 double QwtScaleArithmetic::ceil125 (double *x*) [static]

Find the smallest value out of $\{1,2,5\} \cdot 10^n$ with an integer number n which is greater than or equal to x

Parameters

<i>x</i>	Input value
----------	-------------

12.71.2.2 double QwtScaleArithmetic::ceilEps (double *value*, double *intervalSize*) [static]

Ceil a value, relative to an interval

Parameters

<i>value</i>	Value to ceil
<i>intervalSize</i>	Interval size

See also

[floorEps\(\)](#)

12.71.2.3 int QwtScaleArithmetic::compareEps (double *value1*, double *value2*, double *intervalSize*) [static]

Compare 2 values, relative to an interval.

Values are "equal", when : $-value2 - value1 \leq abs(intervalSize * 10e^{-6})$

Parameters

<i>value1</i>	First value to compare
<i>value2</i>	Second value to compare
<i>intervalSize</i>	interval size

Returns

0: if equal, -1: if value2 > value1, 1: if value1 > value2

12.71.2.4 double QwtScaleArithmetic::divideEps (double *intervalSize*, double *numSteps*) [static]

Divide an interval into steps.

$$stepSize = (intervalSize - intervalSize * 10e^{-6}) / numSteps$$

Parameters

<i>intervalSize</i>	Interval size
<i>numSteps</i>	Number of steps

Returns

Step size

12.71.2.5 double QwtScaleArithmetic::floor125 (double *x*) [static]

Find the largest value out of {1,2,5}*10^n with an integer number n which is smaller than or equal to x.

Parameters

<i>x</i>	Input value
----------	-------------

12.71.2.6 double QwtScaleArithmetic::floorEps (double *value*, double *intervalSize*) [static]

Floor a value, relative to an interval

Parameters

<i>value</i>	Value to floor
<i>intervalSize</i>	Interval size

See also

[floorEps\(\)](#)

12.72 QwtScaleDiv Class Reference

A class representing a scale division.

```
#include <qwt_scale_div.h>
```

Public Types

- enum [TickType](#) {
 NoTick = -1,
 MinorTick,
 MediumTick,
 MajorTick,
 NTickTypes }

Public Member Functions

- bool [contains](#) (double v) const
- [QwtDoubleInterval interval](#) () const
- void [invalidate](#) ()
- void [invert](#) ()
- bool [isValid](#) () const
- double [lowerBound](#) () const
- int [operator!=](#) (const [QwtScaleDiv](#) &s) const
- int [operator==](#) (const [QwtScaleDiv](#) &s) const
- [QwtScaleDiv](#) ()
- [QwtScaleDiv](#) (double lowerBound, double upperBound, [QwtValueList](#)[[NTickTypes](#)])
- [QwtScaleDiv](#) (const [QwtDoubleInterval](#) &, [QwtValueList](#)[[NTickTypes](#)])
- double [range](#) () const
- void [setInterval](#) (const [QwtDoubleInterval](#) &)
- void [setInterval](#) (double lowerBound, double upperBound)
- void [setTicks](#) (int type, const [QwtValueList](#) &)
- const [QwtValueList](#) & [ticks](#) (int type) const
- double [upperBound](#) () const

12.72.1 Detailed Description

A class representing a scale division. A scale division consists of its limits and 3 list of tick values qualified as major, medium and minor ticks.

In most cases scale divisions are calculated by a [QwtScaleEngine](#).

See also

[subDivideInto\(\)](#), [subDivide\(\)](#)

12.72.2 Member Enumeration Documentation

12.72.2.1 enum QwtScaleDiv::TickType

Scale tick types.

12.72.3 Constructor & Destructor Documentation

12.72.3.1 QwtScaleDiv::QwtScaleDiv () [explicit]

Construct an invalid [QwtScaleDiv](#) instance.

12.72.3.2 QwtScaleDiv::QwtScaleDiv (const QwtDoubleInterval & *interval*,
QwtValueList *ticks*[*NTickTypes*]) [explicit]

Construct [QwtScaleDiv](#) instance.

Parameters

<i>interval</i>	Interval
<i>ticks</i>	List of major, medium and minor ticks

12.72.3.3 QwtScaleDiv::QwtScaleDiv (double *lowerBound*, double
upperBound, QwtValueList *ticks*[*NTickTypes*]) [explicit]

Construct [QwtScaleDiv](#) instance.

Parameters

<i>lowerBound</i>	First interval limit
<i>upperBound</i>	Second interval limit
<i>ticks</i>	List of major, medium and minor ticks

12.72.4 Member Function Documentation

12.72.4.1 bool QwtScaleDiv::contains (double *value*) const

Return if a value is between [lowerBound\(\)](#) and [upperBound\(\)](#)

Parameters

<i>value</i>	Value
--------------	-------

Returns

true/false

12.72.4.2 QwtDoubleInterval QwtScaleDiv::interval () const [inline]**Returns**

lowerBound -> upperBound

12.72.4.3 void QwtScaleDiv::invalidate ()

Invalidate the scale division.

12.72.4.4 void QwtScaleDiv::invert ()

Invert the scale division.

12.72.4.5 bool QwtScaleDiv::isValid () const

Check if the scale division is valid.

12.72.4.6 double QwtScaleDiv::lowerBound () const [inline]**Returns**

lower bound

See also

[upperBound\(\)](#)

12.72.4.7 int QwtScaleDiv::operator!= (const QwtScaleDiv & s) const

Inequality.

Returns

true if this instance is not equal to s

12.72.4.8 int QwtScaleDiv::operator==(const QwtScaleDiv & *other*) const

Equality operator.

Returns

true if this instance is equal to other

12.72.4.9 double QwtScaleDiv::range () const [inline]**Returns**

[upperBound\(\)](#) - [lowerBound\(\)](#)

12.72.4.10 void QwtScaleDiv::setInterval (const QwtDoubleInterval & *interval*)

Change the interval

Parameters

<i>interval</i>	Interval
-----------------	----------

12.72.4.11 void QwtScaleDiv::setInterval (double *lowerBound*, double *upperBound*) [inline]

Change the interval

Parameters

<i>lowerBound</i>	lower bound
<i>upperBound</i>	upper bound

12.72.4.12 void QwtScaleDiv::setTicks (int *type*, const QwtValueList & *ticks*)

Assign ticks

Parameters

<i>type</i>	MinorTick, MediumTick or MajorTick
<i>ticks</i>	Values of the tick positions

12.72.4.13 const QwtValueList & QwtScaleDiv::ticks (int *type*) const

Return a list of ticks

Parameters

<i>type</i>	MinorTick, MediumTick or MajorTick
-------------	------------------------------------

12.72.4.14 double QwtScaleDiv::upperBound () const [inline]**Returns**

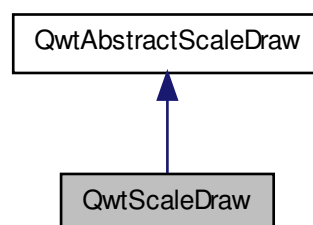
upper bound

See also[lowerBound\(\)](#)**12.73 QwtScaleDraw Class Reference**

A class for drawing scales.

```
#include <qwt_scale_draw.h>
```

Inheritance diagram for QwtScaleDraw:



Public Types

- enum [Alignment](#) {
BottomScale,
TopScale,
LeftScale,
RightScale }
- enum [ScaleComponent](#) {
Backbone = 1,
Ticks = 2,
Labels = 4 }

Public Member Functions

- [Alignment](#) [alignment](#) () const
- QRect [boundingLabelRect](#) (const QFont &, double val) const
- virtual void [draw](#) (QPainter *, const QPalette &) const
- void [enableComponent](#) ([ScaleComponent](#), bool enable=true)
- virtual int [extent](#) (const QPen &, const QFont &) const
- void [getBorderDistHint](#) (const QFont &, int &start, int &end) const
- bool [hasComponent](#) ([ScaleComponent](#)) const
- virtual [QwtText](#) [label](#) (double) const
- Qt::Alignment [labelAlignment](#) () const
- QPoint [labelPosition](#) (double val) const
- QRect [labelRect](#) (const QFont &, double val) const
- double [labelRotation](#) () const
- QSize [labelSize](#) (const QFont &, double val) const
- int [length](#) () const
- int [majTickLength](#) () const
- const [QwtScaleMap](#) & [map](#) () const
- int [maxLabelHeight](#) (const QFont &) const
- int [maxLabelWidth](#) (const QFont &) const
- int [minimumExtent](#) () const
- int [minLabelDist](#) (const QFont &) const
- int [minLength](#) (const QPen &, const QFont &) const
- void [move](#) (int x, int y)
- void [move](#) (const QPoint &)
- [QwtScaleDraw](#) & [operator=](#) (const [QwtScaleDraw](#) &other)
- Qt::Orientation [orientation](#) () const
- QPoint [pos](#) () const
- [QwtScaleDraw](#) (const [QwtScaleDraw](#) &)
- [QwtScaleDraw](#) ()
- const [QwtScaleDiv](#) & [scaleDiv](#) () const
- [QwtScaleMap](#) & [scaleMap](#) ()
- void [setAlignment](#) ([Alignment](#))

- void [setLabelAlignment](#) (Qt::Alignment)
- void [setLabelRotation](#) (double rotation)
- void [setLength](#) (int length)
- void [setMinimumExtent](#) (int)
- void [setScaleDiv](#) (const [QwtScaleDiv](#) &s)
- void [setSpacing](#) (int margin)
- void [setTickLength](#) ([QwtScaleDiv::TickType](#), int length)
- void [setTransformation](#) ([QwtScaleTransformation](#) *)
- int [spacing](#) () const
- int [tickLength](#) ([QwtScaleDiv::TickType](#)) const
- virtual [~QwtScaleDraw](#) ()

Protected Member Functions

- virtual void [drawBackbone](#) (QPainter *p) const
- virtual void [drawLabel](#) (QPainter *p, double val) const
- virtual void [drawTick](#) (QPainter *p, double val, int len) const
- void [invalidateCache](#) ()
- QMatrix [labelMatrix](#) (const QPoint &, const QSize &) const
- const [QwtText](#) & [tickLabel](#) (const QFont &, double value) const

12.73.1 Detailed Description

A class for drawing scales. [QwtScaleDraw](#) can be used to draw linear or logarithmic scales. A scale has a position, an alignment and a length, which can be specified. The labels can be rotated and aligned to the ticks using [setLabelRotation\(\)](#) and [setLabelAlignment\(\)](#).

After a scale division has been specified as a [QwtScaleDiv](#) object using [QwtAbstractScaleDraw::setScaleDiv\(const QwtScaleDiv &s\)](#), the scale can be drawn with the [QwtAbstractScaleDraw::draw\(\)](#) member.

12.73.2 Member Enumeration Documentation

12.73.2.1 enum QwtScaleDraw::Alignment

Alignment of the scale draw

See also

[setAlignment\(\)](#), [alignment\(\)](#)

12.73.2.2 enum QwtAbstractScaleDraw::ScaleComponent [inherited]

Components of a scale

- Backbone
- Ticks
- Labels

See also

[enableComponent\(\)](#), [hasComponent](#)

12.73.3 Constructor & Destructor Documentation

12.73.3.1 QwtScaleDraw::QwtScaleDraw ()

Constructor.

The range of the scale is initialized to [0, 100], The position is at (0, 0) with a length of 100. The orientation is QwtAbstractScaleDraw::Bottom.

12.73.3.2 QwtScaleDraw::QwtScaleDraw (const QwtScaleDraw & *other*)

Copy constructor.

12.73.3.3 QwtScaleDraw::~QwtScaleDraw () [virtual]

Destructor.

12.73.4 Member Function Documentation

12.73.4.1 QwtScaleDraw::Alignment QwtScaleDraw::alignment () const

Return alignment of the scale

See also

[setAlignment\(\)](#)

12.73.4.2 QRect QwtScaleDraw::boundingLabelRect (const QFont & *font*, double *value*) const

Find the bounding rect for the label. The coordinates of the rect are absolute coordinates (calculated from [pos\(\)](#)). in direction of the tick.

Parameters

<i>font</i>	Font used for painting
<i>value</i>	Value

See also

[labelRect\(\)](#)

12.73.4.3 void QwtAbstractScaleDraw::draw (QPainter * *painter*, const QPalette & *palette*) const [virtual, inherited]

Draw the scale.

Parameters

<i>painter</i>	The painter
<i>palette</i>	Palette, text color is used for the labels, foreground color for ticks and backbone

12.73.4.4 void QwtScaleDraw::drawBackbone (QPainter * *painter*) const [protected, virtual]

Draws the baseline of the scale

Parameters

<i>painter</i>	Painter
----------------	---------

See also

[drawTick\(\)](#), [drawLabel\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.73.4.5 void QwtScaleDraw::drawLabel (QPainter * *painter*, double *value*) const [protected, virtual]

Draws the label for a major scale tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value

See also

[drawTick\(\)](#), [drawBackbone\(\)](#), [boundingLabelRect\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.73.4.6 `void QwtScaleDraw::drawTick (QPainter * painter, double value, int len) const` `[protected, virtual]`

Draw a tick

Parameters

<i>painter</i>	Painter
<i>value</i>	Value of the tick
<i>len</i>	Lenght of the tick

See also

[drawBackbone\(\)](#), [drawLabel\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.73.4.7 `void QwtAbstractScaleDraw::enableComponent (ScaleComponent component, bool enable = true)` `[inherited]`

En/Disable a component of the scale

Parameters

<i>component</i>	Scale component
<i>enable</i>	On/Off

See also

[hasComponent\(\)](#)

12.73.4.8 `int QwtScaleDraw::extent (const QPen & pen, const QFont & font) const` `[virtual]`

Calculate the width/height that is needed for a vertical/horizontal scale.

The extent is calculated from the pen width of the backbone, the major tick length, the spacing and the maximum width/height of the labels.

Parameters

<i>pen</i>	Pen that is used for painting backbone and ticks
<i>font</i>	Font used for painting the labels

See also

[minLength\(\)](#)

Implements [QwtAbstractScaleDraw](#).

12.73.4.9 void QwtScaleDraw::getBorderDistHint (const QFont & *font*, int & *start*, int & *end*) const

Determine the minimum border distance.

This member function returns the minimum space needed to draw the mark labels at the scale's endpoints.

Parameters

<i>font</i>	Font
<i>start</i>	Start border distance
<i>end</i>	End border distance

12.73.4.10 bool QwtAbstractScaleDraw::hasComponent (ScaleComponent *component*) const [inherited]

Check if a component is enabled

See also

[enableComponent\(\)](#)

12.73.4.11 void QwtAbstractScaleDraw::invalidateCache () [protected, inherited]

Invalidate the cache used by [QwtAbstractScaleDraw::tickLabel](#)

The cache is invalidated, when a new [QwtScaleDiv](#) is set. If the labels need to be changed. while the same [QwtScaleDiv](#) is set, [QwtAbstractScaleDraw::invalidateCache](#) needs to be called manually.

12.73.4.12 QText QwtAbstractScaleDraw::label (double *value*) const [virtual, inherited]

Convert a value into its representing label.

The value is converted to a plain text using `QLocale::system().toString(value)`. This method is often overloaded by applications to have individual labels.

Parameters

<i>value</i>	Value
--------------	-------

Returns

Label string.

Reimplemented in [QwtDialScaleDraw](#).

12.73.4.13 Qt::Alignment QwtScaleDraw::labelAlignment () const

Returns

the label flags

See also

[setLabelAlignment\(\)](#), [labelRotation\(\)](#)

12.73.4.14 QMatrix QwtScaleDraw::labelMatrix (const QPoint & pos, const QSize & size) const **[protected]**

Calculate the matrix that is needed to paint a label depending on its alignment and rotation.

Parameters

<i>pos</i>	Position where to paint the label
<i>size</i>	Size of the label

See also

[setLabelAlignment\(\)](#), [setLabelRotation\(\)](#)

12.73.4.15 QPoint QwtScaleDraw::labelPosition (double value) const

Find the position, where to paint a label

The position has a distance of [majTickLength\(\)](#) + [spacing\(\)](#) + 1 from the backbone. The direction depends on the [alignment\(\)](#)

Parameters

<i>value</i>	Value
--------------	-------

12.73.4.16 QRect QwtScaleDraw::labelRect (const QFont & *font*, double *value*) const

Find the bounding rect for the label. The coordinates of the rect are relative to spacing + ticklength from the backbone in direction of the tick.

Parameters

<i>font</i>	Font used for painting
<i>value</i>	Value

12.73.4.17 double QwtScaleDraw::labelRotation () const**Returns**

the label rotation

See also

[setLabelRotation\(\)](#), [labelAlignment\(\)](#)

12.73.4.18 QSize QwtScaleDraw::labelSize (const QFont & *font*, double *value*) const

Calculate the size that is needed to draw a label

Parameters

<i>font</i>	Label font
<i>value</i>	Value

12.73.4.19 int QwtScaleDraw::length () const**Returns**

the length of the backbone

See also

[setLength\(\)](#), [pos\(\)](#)

12.73.4.20 `int QwtAbstractScaleDraw::majTickLength () const`
[*inherited*]

The same as `QwtAbstractScaleDraw::tickLength(QwtScaleDiv::MajorTick)`.

12.73.4.21 `const QwtScaleMap & QwtAbstractScaleDraw::map () const`
[*inherited*]

Returns

Map how to translate between scale and pixel values

12.73.4.22 `int QwtScaleDraw::maxLabelHeight (const QFont & font) const`

Parameters

<i>font</i>	Font
-------------	------

Returns

the maximum height of a label

12.73.4.23 `int QwtScaleDraw::maxLabelWidth (const QFont & font) const`

Parameters

<i>font</i>	Font
-------------	------

Returns

the maximum width of a label

12.73.4.24 `int QwtAbstractScaleDraw::minimumExtent () const`
[*inherited*]

Get the minimum extent

See also

[extent\(\)](#), [setMinimumExtent\(\)](#)

12.73.4.25 int QwtScaleDraw::minLabelDist (const QFont & *font*) const

Determine the minimum distance between two labels, that is necessary that the texts don't overlap.

Parameters

<i>font</i>	Font
-------------	------

Returns

The maximum width of a label

See also

[getBorderDistHint\(\)](#)

12.73.4.26 int QwtScaleDraw::minLength (const QPen & *pen*, const QFont & *font*) const

Calculate the minimum length that is needed to draw the scale

Parameters

<i>pen</i>	Pen that is used for painting backbone and ticks
<i>font</i>	Font used for painting the labels

See also

[extent\(\)](#)

12.73.4.27 void QwtScaleDraw::move (int *x*, int *y*) [inline]

Move the position of the scale

See also

[move\(const QPoint &\)](#)

12.73.4.28 void QwtScaleDraw::move (const QPoint & *pos*)

Move the position of the scale.

The meaning of the parameter *pos* depends on the alignment:

QwtScaleDraw::LeftScale The origin is the topmost point of the backbone. The backbone is a vertical line. Scale marks and labels are drawn at the left of the backbone.

QwtScaleDraw::RightScale The origin is the topmost point of the backbone. The backbone is a vertical line. Scale marks and labels are drawn at the right of the backbone.

QwtScaleDraw::TopScale The origin is the leftmost point of the backbone. The backbone is a horizontal line. Scale marks and labels are drawn above the backbone.

QwtScaleDraw::BottomScale The origin is the leftmost point of the backbone. The backbone is a horizontal line. Scale marks and labels are drawn below the backbone.

Parameters

<i>pos</i>	Origin of the scale
------------	---------------------

See also

[pos\(\)](#), [setLength\(\)](#)

12.73.4.29 QwtScaleDraw & QwtScaleDraw::operator= (const QwtScaleDraw & *other*)

Assignment operator.

12.73.4.30 Qt::Orientation QwtScaleDraw::orientation () const

Return the orientation

TopScale, BottomScale are horizontal (Qt::Horizontal) scales, LeftScale, RightScale are vertical (Qt::Vertical) scales.

See also

[alignment\(\)](#)

12.73.4.31 QPoint QwtScaleDraw::pos () const

Returns

Origin of the scale

See also

[move\(\)](#), [length\(\)](#)

12.73.4.32 `const QwtScaleDiv & QwtAbstractScaleDraw::scaleDiv () const`
[inherited]

Returns

scale division

12.73.4.33 `QwtScaleMap & QwtAbstractScaleDraw::scaleMap ()`
[inherited]

Returns

Map how to translate between scale and pixel values

12.73.4.34 `void QwtScaleDraw::setAlignment (Alignment align)`

Set the alignment of the scale

The default alignment is QwtScaleDraw::BottomScale

See also

[alignment\(\)](#)

12.73.4.35 `void QwtScaleDraw::setLabelAlignment (Qt::Alignment alignment)`

Change the label flags.

Labels are aligned to the point ticklength + spacing away from the backbone.

The alignment is relative to the orientation of the label text. In case of an flags of 0 the label will be aligned depending on the orientation of the scale:

QwtScaleDraw::TopScale: Qt::AlignHCenter | Qt::AlignTop

QwtScaleDraw::BottomScale: Qt::AlignHCenter | Qt::AlignBottom

QwtScaleDraw::LeftScale: Qt::AlignLeft | Qt::AlignVCenter

QwtScaleDraw::RightScale: Qt::AlignRight | Qt::AlignVCenter

Changing the alignment is often necessary for rotated labels.

Parameters

<i>alignment</i>	Or'd Qt::AlignmentFlags <see qnamespace.h>
------------------	--

See also

[setLabelRotation\(\)](#), [labelRotation\(\)](#), [labelAlignment\(\)](#)

Warning

The various alignments might be confusing. The alignment of the label is not the alignment of the scale and is not the alignment of the flags ([QwtText::flags\(\)](#)) returned from [QwtAbstractScaleDraw::label\(\)](#).

12.73.4.36 void QwtScaleDraw::setLabelRotation (double *rotation*)

Rotate all labels.

When changing the rotation, it might be necessary to adjust the label flags too. Finding a useful combination is often the result of try and error.

Parameters

<i>rotation</i>	Angle in degrees. When changing the label rotation, the label flags often needs to be adjusted too.
-----------------	---

See also

[setLabelAlignment\(\)](#), [labelRotation\(\)](#), [labelAlignment\(\)](#).

12.73.4.37 void QwtScaleDraw::setLength (int *length*)

Set the length of the backbone.

The length doesn't include the space needed for overlapping labels.

See also

[move\(\)](#), [minLabelDist\(\)](#)

**12.73.4.38 void QwtAbstractScaleDraw::setMinimumExtent (int *minExtent*)
[inherited]**

Set a minimum for the extent.

The extent is calculated from the components of the scale draw. In situations, where the labels are changing and the layout depends on the extent (f.e scrolling a scale), setting an upper limit as minimum extent will avoid jumps of the layout.

Parameters

<i>minExtent</i>	Minimum extent
------------------	----------------

See also

[extent\(\)](#), [minimumExtent\(\)](#)

12.73.4.39 void QwtAbstractScaleDraw::setScaleDiv (const QwtScaleDiv & *sd*) **[inherited]**

Change the scale division

Parameters

<i>sd</i>	New scale division
-----------	--------------------

12.73.4.40 void QwtAbstractScaleDraw::setSpacing (int *spacing*) **[inherited]**

Set the spacing between tick and labels.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

[spacing\(\)](#)

12.73.4.41 void QwtAbstractScaleDraw::setTickLength (QwtScaleDiv::TickType *tickType*, int *length*) **[inherited]**

Set the length of the ticks

Parameters

<i>tickType</i>	Tick type
<i>length</i>	New length

Warning

the length is limited to [0..1000]

12.73.4.42 void QwtAbstractScaleDraw::setTransformation (QwtScaleTransformation * *transformation*) **[inherited]**

Change the transformation of the scale

Parameters

<i>transformation</i>	New scale transformation
-----------------------	--------------------------

12.73.4.43 int QwtAbstractScaleDraw::spacing () const [inherited]

Get the spacing.

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

See also

[setSpacing\(\)](#)

12.73.4.44 const QwtText & QwtAbstractScaleDraw::tickLabel (const QFont & font, double value) const [protected, inherited]

Convert a value into its representing label and cache it.

The conversion between value and label is called very often in the layout and painting code. Unfortunately the calculation of the label sizes might be slow (really slow for rich text in Qt4), so it's necessary to cache the labels.

Parameters

<i>font</i>	Font
<i>value</i>	Value

Returns

Tick label

12.73.4.45 int QwtAbstractScaleDraw::tickLength (QwtScaleDiv::TickType tickType) const [inherited]

Return the length of the ticks

See also

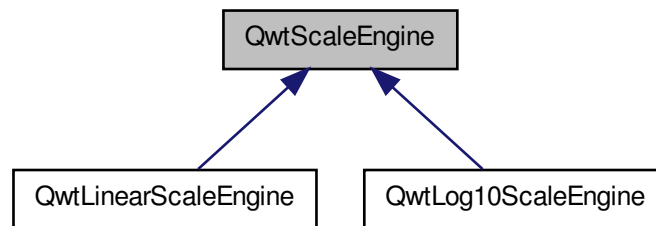
[setTickLength\(\)](#), [majTickLength\(\)](#)

12.74 QwtScaleEngine Class Reference

Base class for scale engines.

```
#include <qwt_scale_engine.h>
```

Inheritance diagram for QwtScaleEngine:



Public Types

- enum [Attribute](#) {
NoAttribute = 0,
IncludeReference = 1,
Symmetric = 2,
Floating = 4,
Inverted = 8 }

Public Member Functions

- int [attributes](#) () const
- virtual void [autoScale](#) (int maxNumSteps, double &x1, double &x2, double &stepSize) const =0
- virtual [QwtScaleDiv](#) [divideScale](#) (double x1, double x2, int maxMajSteps, int maxMinSteps, double stepSize=0.0) const =0
- double [lowerMargin](#) () const
- [QwtScaleEngine](#) ()
- double [reference](#) () const
- void [setAttribute](#) ([Attribute](#), bool on=true)
- void [setAttributes](#) (int)
- void [setMargins](#) (double lower, double upper)
- void [setReference](#) (double reference)
- bool [testAttribute](#) ([Attribute](#)) const
- virtual [QwtScaleTransformation](#) * [transformation](#) () const =0
- double [upperMargin](#) () const
- virtual [~QwtScaleEngine](#) ()

Protected Member Functions

- [QwtDoubleInterval buildInterval](#) (double v) const
- bool [contains](#) (const [QwtDoubleInterval](#) &, double val) const
- double [divideInterval](#) (double interval, int numSteps) const
- [QwtValueList strip](#) (const [QwtValueList](#) &, const [QwtDoubleInterval](#) &) const

12.74.1 Detailed Description

Base class for scale engines. A scale engine tries to find "reasonable" ranges and step sizes for scales.

The layout of the scale can be varied with [setAttribute\(\)](#).

Qwt offers implementations for logarithmic (log10) and linear scales. Contributions for other types of scale engines (date/time, log2 ...) are welcome.

12.74.2 Member Enumeration Documentation

12.74.2.1 enum QwtScaleEngine::Attribute

- IncludeReference
Build a scale which includes the [reference\(\)](#) value.
- Symmetric
Build a scale which is symmetric to the [reference\(\)](#) value.
- Floating
The endpoints of the scale are supposed to be equal the outmost included values plus the specified margins (see [setMargins\(\)](#)). If this attribute is **not** set, the endpoints of the scale will be integer multiples of the step size.
- Inverted
Turn the scale upside down.

See also

[setAttribute\(\)](#), [testAttribute\(\)](#), [reference\(\)](#), [lowerMargin\(\)](#), [upperMargin\(\)](#)

12.74.3 Constructor & Destructor Documentation

12.74.3.1 QwtScaleEngine::QwtScaleEngine () [explicit]

Constructor.

12.74.3.2 QwtScaleEngine::~~QwtScaleEngine () [virtual]

Destructor.

12.74.4 Member Function Documentation**12.74.4.1 int QwtScaleEngine::attributes () const**

Return the scale attributes

See also

[Attribute](#), [setAttributes\(\)](#), [testAttribute\(\)](#)

12.74.4.2 virtual void QwtScaleEngine::autoScale (int *maxNumSteps*, double & *x1*, double & *x2*, double & *stepSize*) const [pure virtual]

Align and divide an interval

Parameters

<i>maxNumSteps</i>	Max. number of steps
<i>x1</i>	First limit of the interval (In/Out)
<i>x2</i>	Second limit of the interval (In/Out)
<i>stepSize</i>	Step size (Return value)

Implemented in [QwtLinearScaleEngine](#), and [QwtLog10ScaleEngine](#).

12.74.4.3 QwtDoubleInterval QwtScaleEngine::buildInterval (double *v*) const [protected]

Build an interval for a value.

In case of $v == 0.0$ the interval is $[-0.5, 0.5]$, otherwise it is $[0.5 * v, 1.5 * v]$

12.74.4.4 bool QwtScaleEngine::contains (const QwtDoubleInterval & *interval*, double *value*) const [protected]

Check if an interval "contains" a value

Parameters

<i>interval</i>	Interval
<i>value</i>	Value

See also

[QwtScaleArithmetic::compareEps\(\)](#)

12.74.4.5 `double QwtScaleEngine::divideInterval (double intervalSize, int numSteps) const` **[protected]**

Calculate a step size for an interval size

Parameters

<i>intervalSize</i>	Interval size
<i>numSteps</i>	Number of steps

Returns

Step size

12.74.4.6 `virtual QwtScaleDiv QwtScaleEngine::divideScale (double x1, double x2, int maxMajSteps, int maxMinSteps, double stepSize = 0.0) const` **[pure virtual]**

Calculate a scale division.

Parameters

<i>x1</i>	First interval limit
<i>x2</i>	Second interval limit
<i>maxMajSteps</i>	Maximum for the number of major steps
<i>maxMinSteps</i>	Maximum number of minor steps
<i>stepSize</i>	Step size. If <i>stepSize</i> == 0.0, the scaleEngine calculates one.

Implemented in [QwtLinearScaleEngine](#), and [QwtLog10ScaleEngine](#).

12.74.4.7 `double QwtScaleEngine::lowerMargin () const`

Returns

the margin at the lower end of the scale The default margin is 0.

See also

[setMargins\(\)](#)

12.74.4.8 double QwtScaleEngine::reference () const**Returns**

the reference value

See also

[setReference\(\)](#), [setAttribute\(\)](#)

12.74.4.9 void QwtScaleEngine::setAttribute (Attribute *attribute*, bool *on* = *true*)

Change a scale attribute

Parameters

<i>attribute</i>	Attribute to change
<i>on</i>	On/Off

See also

[Attribute](#), [testAttribute\(\)](#)

12.74.4.10 void QwtScaleEngine::setAttributes (int *attributes*)

Change the scale attribute

Parameters

<i>attributes</i>	Set scale attributes
-------------------	----------------------

See also

[Attribute](#), [attributes\(\)](#)

12.74.4.11 void QwtScaleEngine::setMargins (double *lower*, double *upper*)

Specify margins at the scale's endpoints.

Parameters

<i>lower</i>	minimum distance between the scale's lower boundary and the smallest enclosed value
<i>upper</i>	minimum distance between the scale's upper boundary and the greatest enclosed value

Margins can be used to leave a minimum amount of space between the enclosed intervals and the boundaries of the scale.

Warning

- [QwtLog10ScaleEngine](#) measures the margins in decades.

See also

[upperMargin\(\)](#), [lowerMargin\(\)](#)

12.74.4.12 void QwtScaleEngine::setReference (double *r*)

Specify a reference point.

Parameters

<i>r</i>	new reference value
----------	---------------------

The reference point is needed if options IncludeReference or Symmetric are active. Its default value is 0.0.

See also

[Attribute](#)

12.74.4.13 QwtValueList QwtScaleEngine::strip (const QwtValueList & *ticks*, const QwtDoubleInterval & *interval*) const [protected]

Remove ticks from a list, that are not inside an interval

Parameters

<i>ticks</i>	Tick list
<i>interval</i>	Interval

Returns

Stripped tick list

12.74.4.14 bool QwtScaleEngine::testAttribute (Attribute *attribute*) const

Check if a attribute is set.

Parameters

<i>attribute</i>	Attribute to be tested
------------------	------------------------

See also

[Attribute](#), [setAttribute\(\)](#)

12.74.4.15 `virtual QwtScaleTransformation* QwtScaleEngine::transformation
() const [pure virtual]`

Returns

a transformation

Implemented in [QwtLinearScaleEngine](#), and [QwtLog10ScaleEngine](#).

12.74.4.16 `double QwtScaleEngine::upperMargin () const`

Returns

the margin at the upper end of the scale The default margin is 0.

See also

[setMargins\(\)](#)

12.75 QwtScaleMap Class Reference

A scale map.

```
#include <qwt_scale_map.h>
```

Public Member Functions

- double [invTransform](#) (double i) const
- [QwtScaleMap](#) & [operator=](#) (const [QwtScaleMap](#) &)
- double [p1](#) () const
- double [p2](#) () const
- double [pDist](#) () const
- [QwtScaleMap](#) (const [QwtScaleMap](#) &)
- [QwtScaleMap](#) ()
- double [s1](#) () const
- double [s2](#) () const
- double [sDist](#) () const
- void [setPaintInterval](#) (int p1, int p2)
- void [setPaintXInterval](#) (double p1, double p2)
- void [setScaleInterval](#) (double s1, double s2)

- void [setTransformation](#) ([QwtScaleTransformation](#) *)
- int [transform](#) (double x) const
- const [QwtScaleTransformation](#) * [transformation](#) () const
- double [xTransform](#) (double x) const
- [~QwtScaleMap](#) ()

Public Attributes

- QT_STATIC_CONST double **LogMax** = 1.0e150
- QT_STATIC_CONST double **LogMin** = 1.0e-150

12.75.1 Detailed Description

A scale map. [QwtScaleMap](#) offers transformations from a scale into a paint interval and vice versa.

12.75.2 Constructor & Destructor Documentation

12.75.2.1 [QwtScaleMap::QwtScaleMap](#) ()

Constructor.

The scale and paint device intervals are both set to [0,1].

12.75.2.2 [QwtScaleMap::QwtScaleMap](#) (const [QwtScaleMap](#) & *other*)

Copy constructor.

12.75.2.3 [QwtScaleMap::~~QwtScaleMap](#) ()

Destructor

12.75.3 Member Function Documentation

12.75.3.1 double [QwtScaleMap::invTransform](#) (double *p*) const [\[inline\]](#)

Transform an paint device value into a value in the interval of the scale.

Parameters

<i>p</i>	Value relative to the coordinates of the paint device
----------	---

See also

[transform\(\)](#)

12.75.3.2 QwtScaleMap & QwtScaleMap::operator= (const QwtScaleMap & *other*)

Assignment operator.

12.75.3.3 double QwtScaleMap::p1 () const [inline]

Returns

First border of the paint interval

12.75.3.4 double QwtScaleMap::p2 () const [inline]

Returns

Second border of the paint interval

12.75.3.5 double QwtScaleMap::pDist () const [inline]

Returns

[qwtAbs\(p2\(\) - p1\(\)\)](#)

12.75.3.6 double QwtScaleMap::s1 () const [inline]

Returns

First border of the scale interval

12.75.3.7 double QwtScaleMap::s2 () const [inline]

Returns

Second border of the scale interval

12.75.3.8 double QwtScaleMap::sDist () const [inline]**Returns**

qwtAbs(s2() - s1())

12.75.3.9 void QwtScaleMap::setPaintInterval (int p1, int p2)

Specify the borders of the paint device interval.

Parameters

<i>p1</i>	first border
<i>p2</i>	second border

12.75.3.10 void QwtScaleMap::setPaintXInterval (double p1, double p2)

Specify the borders of the paint device interval.

Parameters

<i>p1</i>	first border
<i>p2</i>	second border

12.75.3.11 void QwtScaleMap::setScaleInterval (double s1, double s2)

Specify the borders of the scale interval.

Parameters

<i>s1</i>	first border
<i>s2</i>	second border

Warning

logarithmic scales might be aligned to [LogMin, LogMax]

12.75.3.12 `void QwtScaleMap::setTransformation (QwtScaleTransformation * transformation)`

Initialize the map with a transformation

12.75.3.13 `int QwtScaleMap::transform (double s) const [inline]`

Transform a point related to the scale interval into an point related to the interval of the paint device and round it to an integer. (In Qt <= 3.x paint devices are integer based.)

Parameters

<i>s</i>	Value relative to the coordinates of the scale
----------	--

See also

[xTransform\(\)](#)

12.75.3.14 `const QwtScaleTransformation * QwtScaleMap::transformation () const`

Get the transformation.

12.75.3.15 `double QwtScaleMap::xTransform (double s) const [inline]`

Transform a point related to the scale interval into an point related to the interval of the paint device

Parameters

<i>s</i>	Value relative to the coordinates of the scale
----------	--

12.76 QwtScaleTransformation Class Reference

Operations for linear or logarithmic (base 10) transformations.

```
#include <qwt_scale_map.h>
```

Public Types

- enum **Type** {
Linear,
Log10,
Other }

Public Member Functions

- virtual [QwtScaleTransformation](#) * [copy](#) () const
- virtual double [invXForm](#) (double x, double p1, double p2, double s1, double s2) const
- [QwtScaleTransformation](#) (Type type)
- Type [type](#) () const
- virtual double [xForm](#) (double x, double s1, double s2, double p1, double p2) const
- virtual [~QwtScaleTransformation](#) ()

12.76.1 Detailed Description

Operations for linear or logarithmic (base 10) transformations.

12.76.2 Constructor & Destructor Documentation

12.76.2.1 QwtScaleTransformation::QwtScaleTransformation (Type *type*)

Constructor for a linear transformation.

12.76.2.2 QwtScaleTransformation::~~QwtScaleTransformation () [virtual]

Destructor.

12.76.3 Member Function Documentation

12.76.3.1 QwtScaleTransformation * QwtScaleTransformation::copy () const [virtual]

Create a clone of the transformation.

12.76.3.2 double QwtScaleTransformation::invXForm (double *p*, double *p1*, double *p2*, double *s1*, double *s2*) const [virtual]

Transform a value from the coordinate system of the paint device into the coordinate system of a scale.

Parameters

<i>p</i>	Value related to the coordinate system of the paint device
<i>p1</i>	First border of the coordinate system of the paint device
<i>p2</i>	Second border of the coordinate system of the paint device
<i>s1</i>	First border of the coordinate system of the scale
<i>s2</i>	Second border of the coordinate system of the scale

Returns

linear mapping: $s1 + (s2 - s1) / (p2 - p1) * (p - p1)$;

log10 mapping: $\exp((p - p1) / (p2 - p1) * \log(s2 / s1)) * s1$;

12.76.3.3 QwtScaleTransformation::Type QwtScaleTransformation::type () const [inline]

Returns

Transformation type

12.76.3.4 double QwtScaleTransformation::xForm (double s, double s1, double s2, double p1, double p2) const [virtual]

Transform a value from the coordinate system of a scale into the coordinate system of the paint device.

Parameters

<i>s</i>	Value related to the coordinate system of the scale
<i>s1</i>	First border of the coordinate system of the scale
<i>s2</i>	Second border of the coordinate system of the scale
<i>p1</i>	First border of the coordinate system of the paint device
<i>p2</i>	Second border of the coordinate system of the paint device

Returns

linear mapping: $p1 + (p2 - p1) / (s2 - s1) * (s - s1)$;

log10 mapping: $p1 + (p2 - p1) / \log(s2 / s1) * \log(s / s1)$;

12.77 QwtScaleWidget Class Reference

A Widget which contains a scale.

```
#include <qwt_scale_widget.h>
```

Signals

- void [scaleDivChanged](#) ()

Public Member Functions

- [QwtScaleDraw::Alignment](#) [alignment](#) () const
- [QwtDoubleInterval](#) [colorBarInterval](#) () const
- [QRect](#) [colorBarRect](#) (const [QRect](#) &) const
- int [colorBarWidth](#) () const
- const [QwtColorMap](#) & [colorMap](#) () const
- int [dimForLength](#) (int length, const [QFont](#) &scaleFont) const
- void [drawColorBar](#) ([QPainter](#) *painter, const [QRect](#) &rect) const
- void [drawTitle](#) ([QPainter](#) *painter, [QwtScaleDraw::Alignment](#), const [QRect](#) &rect) const
- int [endBorderDist](#) () const
- void [getBorderDistHint](#) (int &start, int &end) const
- void [getMinBorderDist](#) (int &start, int &end) const
- bool [isColorBarEnabled](#) () const
- int [margin](#) () const
- virtual [QSize](#) [minimumSizeHint](#) () const
- int [penWidth](#) () const
- [QwtScaleWidget](#) ([QWidget](#) *parent=NULL)
- [QwtScaleWidget](#) ([QwtScaleDraw::Alignment](#), [QWidget](#) *parent=NULL)
- const [QwtScaleDraw](#) * [scaleDraw](#) () const
- [QwtScaleDraw](#) * [scaleDraw](#) ()
- void [setAlignment](#) ([QwtScaleDraw::Alignment](#))
- void [setBorderDist](#) (int start, int end)
- void [setColorBarEnabled](#) (bool)
- void [setColorBarWidth](#) (int)
- void [setColorMap](#) (const [QwtDoubleInterval](#) &, const [QwtColorMap](#) &)
- void [setLabelAlignment](#) ([Qt::Alignment](#))
- void [setLabelRotation](#) (double rotation)
- void [setMargin](#) (int)
- void [setMinBorderDist](#) (int start, int end)
- void [setPenWidth](#) (int)
- void [setScaleDiv](#) ([QwtScaleTransformation](#) *, const [QwtScaleDiv](#) &sd)
- void [setScaleDraw](#) ([QwtScaleDraw](#) *)
- void [setSpacing](#) (int td)
- void [setTitle](#) (const [QString](#) &title)
- void [setTitle](#) (const [QwtText](#) &title)
- virtual [QSize](#) [sizeHint](#) () const
- int [spacing](#) () const
- int [startBorderDist](#) () const
- [QwtText](#) [title](#) () const
- int [titleHeightForWidth](#) (int width) const
- virtual [~QwtScaleWidget](#) ()

Protected Member Functions

- void [draw](#) (QPainter *p) const
- void [layoutScale](#) (bool update=true)
- virtual void [paintEvent](#) (QPaintEvent *e)
- virtual void [resizeEvent](#) (QResizeEvent *e)
- void [scaleChange](#) ()

12.77.1 Detailed Description

A Widget which contains a scale. This Widget can be used to decorate composite widgets with a scale.

12.77.2 Constructor & Destructor Documentation
**12.77.2.1 QwtScaleWidget::QwtScaleWidget (QWidget * *parent* = *NULL*)
[explicit]**

Create a scale with the position QwtScaleWidget::Left.

Parameters

<i>parent</i>	Parent widget
---------------	---------------

**12.77.2.2 QwtScaleWidget::QwtScaleWidget (QwtScaleDraw::Alignment
align, QWidget * *parent* = *NULL*) [explicit]**

Constructor.

Parameters

<i>align</i>	Alignment.
<i>parent</i>	Parent widget

12.77.2.3 QwtScaleWidget::~~QwtScaleWidget () [virtual]

Destructor.

12.77.3 Member Function Documentation

12.77.3.1 QwtScaleDraw::Alignment QwtScaleWidget::alignment () const

Returns

position

See also

setPosition()

12.77.3.2 int QwtScaleWidget::dimForLength (int *length*, const QFont & *scaleFont*) const

Find the minimum dimension for a given length. dim is the height, length the width seen in direction of the title.

Parameters

<i>length</i>	width for horizontal, height for vertical scales
<i>scaleFont</i>	Font of the scale

Returns

height for horizontal, width for vertical scales

12.77.3.3 void QwtScaleWidget::draw (QPainter * *p*) const [protected]

draw the scale

12.77.3.4 void QwtScaleWidget::drawTitle (QPainter * *painter*, QwtScaleDraw::Alignment *align*, const QRect & *rect*) const

Rotate and paint a title according to its position into a given rectangle.

Parameters

<i>painter</i>	Painter
<i>align</i>	Alignment
<i>rect</i>	Bounding rectangle

12.77.3.5 int QwtScaleWidget::endBorderDist () const**Returns**

end border distance

See also

[setBorderDist\(\)](#)

12.77.3.6 void QwtScaleWidget::getBorderDistHint (int & start, int & end) const

Calculate a hint for the border distances.

This member function calculates the distance of the scale's endpoints from the widget borders which is required for the mark labels to fit into the widget. The maximum of this distance and the minimum border distance is returned.

Warning

- The minimum border distance depends on the font.

See also

[setMinBorderDist\(\)](#), [getMinBorderDist\(\)](#), [setBorderDist\(\)](#)

12.77.3.7 void QwtScaleWidget::getMinBorderDist (int & start, int & end) const

Get the minimum value for the distances of the scale's endpoints from the widget borders.

See also

[setMinBorderDist\(\)](#), [getBorderDistHint\(\)](#)

12.77.3.8 void QwtScaleWidget::layoutScale (bool update = true) [protected]

Recalculate the scale's geometry and layout based on.

12.77.3.9 int QwtScaleWidget::margin () const**Returns**

margin

See also

[setMargin\(\)](#)

12.77.3.10 QSize QwtScaleWidget::minimumSizeHint () const [virtual]**Returns**

a minimum size hint

**12.77.3.11 void QwtScaleWidget::paintEvent (QPaintEvent * *e*)
[protected, virtual]**

paintEvent

12.77.3.12 int QwtScaleWidget::penWidth () const**Returns**

Scale pen width

See also

[setPenWidth\(\)](#)

**12.77.3.13 void QwtScaleWidget::resizeEvent (QResizeEvent * *e*)
[protected, virtual]**

resizeEvent

12.77.3.14 void QwtScaleWidget::scaleChange () [protected]

Notify a change of the scale.

This virtual function can be overloaded by derived classes. The default implementation updates the geometry and repaints the widget.

12.77.3.15 void QwtScaleWidget::scaleDivChanged () [signal]

Signal emitted, whenever the scale division changes.

12.77.3.16 const QwtScaleDraw * QwtScaleWidget::scaleDraw () const

scaleDraw of this scale

See also

[setScaleDraw\(\)](#), [QwtScaleDraw::setScaleDraw\(\)](#)

12.77.3.17 QwtScaleDraw * QwtScaleWidget::scaleDraw ()

scaleDraw of this scale

See also

[QwtScaleDraw::setScaleDraw\(\)](#)

12.77.3.18 void QwtScaleWidget::setAlignment (QwtScaleDraw::Alignment *alignment*)

Change the alignment

Parameters

<i>alignment</i>	New alignment
------------------	---------------

See also

[alignment\(\)](#)

12.77.3.19 void QwtScaleWidget::setBorderDist (int *dist1*, int *dist2*)

Specify distances of the scale's endpoints from the widget's borders. The actual borders will never be less than minimum border distance.

Parameters

<i>dist1</i>	Left or top Distance
<i>dist2</i>	Right or bottom distance

See also

`borderDist()`

12.77.3.20 void QwtScaleWidget::setLabelAlignment (Qt::Alignment *alignment*)

Change the alignment for the labels.

See also

[QwtScaleDraw::setLabelAlignment\(\)](#), [setLabelRotation\(\)](#)

12.77.3.21 void QwtScaleWidget::setLabelRotation (double *rotation*)

Change the rotation for the labels. See [QwtScaleDraw::setLabelRotation\(\)](#).

Parameters

<i>rotation</i>	Rotation
-----------------	----------

See also

[QwtScaleDraw::setLabelRotation\(\)](#), [setLabelFlags\(\)](#)

12.77.3.22 void QwtScaleWidget::setMargin (int *margin*)

Specify the margin to the colorBar/base line.

Parameters

<i>margin</i>	Margin
---------------	--------

See also

[margin\(\)](#)

12.77.3.23 void QwtScaleWidget::setMinBorderDist (int *start*, int *end*)

Set a minimum value for the distances of the scale's endpoints from the widget borders. This is useful to avoid that the scales are "jumping", when the tick labels or their positions change often.

Parameters

<i>start</i>	Minimum for the start border
<i>end</i>	Minimum for the end border

See also

[getMinBorderDist\(\)](#), [getBorderDistHint\(\)](#)

12.77.3.24 void QwtScaleWidget::setPenWidth (int *width*)

Specify the width of the scale pen.

Parameters

<i>width</i>	Pen width
--------------	-----------

See also

[penWidth\(\)](#)

12.77.3.25 void QwtScaleWidget::setScaleDiv (QwtScaleTransformation * *transformation*, const QwtScaleDiv & *scaleDiv*)

Assign a scale division.

The scale division determines where to set the tick marks.

Parameters

<i>transformation</i>	Transformation, needed to translate between scale and pixal values
<i>scaleDiv</i>	Scale Division

See also

For more information about scale divisions, see [QwtScaleDiv](#).

12.77.3.26 void QwtScaleWidget::setScaleDraw (QwtScaleDraw * *sd*)

Set a scale draw *sd* has to be created with new and will be deleted in [~QwtScaleWidget\(\)](#) or the next call of [setScaleDraw\(\)](#).

Parameters

<i>sd</i>	ScaleDraw object
-----------	------------------

See also

[scaleDraw\(\)](#)

12.77.3.27 void QwtScaleWidget::setSpacing (int *spacing*)

Specify the distance between color bar, scale and title.

Parameters

<i>spacing</i>	Spacing
----------------	---------

See also

[spacing\(\)](#)

12.77.3.28 void QwtScaleWidget::setTitle (const QString & *title*)

Give title new text contents

Parameters

<i>title</i>	New title
--------------	-----------

See also

[title\(\)](#), [setTitle\(const QwtText &\);](#)

12.77.3.29 void QwtScaleWidget::setTitle (const QwtText & *title*)

Give title new text contents

Parameters

<i>title</i>	New title
--------------	-----------

See also

[title\(\)](#)

Warning

The title flags are interpreted in direction of the label, AlignTop, AlignBottom can't be set as the title will always be aligned to the scale.

12.77.3.30 QSize QwtScaleWidget::sizeHint () const [virtual]**Returns**

a size hint

12.77.3.31 int QwtScaleWidget::spacing () const**Returns**

distance between scale and title

See also

[setMargin\(\)](#)

12.77.3.32 int QwtScaleWidget::startBorderDist () const**Returns**

start border distance

See also

[setBorderDist\(\)](#)

12.77.3.33 QwtText QwtScaleWidget::title () const**Returns**

title

See also

[setTitle\(\)](#)

12.77.3.34 int QwtScaleWidget::titleHeightForWidth (int *width*) const

Find the height of the title for a given width.

Parameters

<i>width</i>	Width
--------------	-------

Returns

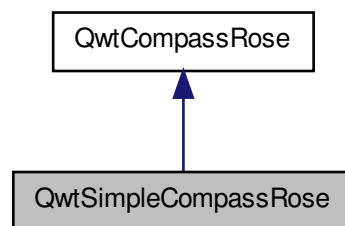
height Height

12.78 QwtSimpleCompassRose Class Reference

A simple rose for [QwtCompass](#).

```
#include <qwt_compass_rose.h>
```

Inheritance diagram for QwtSimpleCompassRose:

**Public Member Functions**

- virtual void [draw](#) (QPainter *, const QPoint ¢er, int radius, double north, QPalette::ColorGroup=QPalette::Active) const
- int [numThornLevels](#) () const
- int [numThorns](#) () const
- const QPalette & [palette](#) () const
- [QwtSimpleCompassRose](#) (int numThorns=8, int numThornLevels=-1)
- void [setNumThornLevels](#) (int count)
- void [setNumThorns](#) (int count)
- virtual void [setPalette](#) (const QPalette &p)
- void [setShrinkFactor](#) (double factor)

- void [setWidth](#) (double w)
- double [shrinkFactor](#) () const
- double [width](#) () const

Static Public Member Functions

- static void [drawRose](#) (QPainter *, const QPalette &, const QPoint ¢er, int radius, double origin, double width, int numThorns, int numThornLevels, double shrinkFactor)

12.78.1 Detailed Description

A simple rose for [QwtCompass](#).

12.78.2 Constructor & Destructor Documentation

12.78.2.1 `QwtSimpleCompassRose::QwtSimpleCompassRose (int numThorns = 8, int numThornLevels = -1)`

Constructor

Parameters

<i>numThorns</i>	Number of thorns
<i>numThornLevels</i>	Number of thorn levels

12.78.3 Member Function Documentation

12.78.3.1 `void QwtSimpleCompassRose::draw (QPainter * painter, const QPoint & center, int radius, double north, QPalette::ColorGroup cg = QPalette::Active) const [virtual]`

Draw the rose

Parameters

<i>painter</i>	Painter
<i>center</i>	Center point
<i>radius</i>	Radius of the rose
<i>north</i>	Position
<i>cg</i>	Color group

Implements [QwtCompassRose](#).

12.78.3.2 void QwtSimpleCompassRose::drawRose (QPainter * *painter*, const QPalette & *palette*, const QPoint & *center*, int *radius*, double *north*, double *width*, int *numThorns*, int *numThornLevels*, double *shrinkFactor*) [static]

Draw the rose

Parameters

<i>painter</i>	Painter
<i>palette</i>	Palette
<i>center</i>	Center of the rose
<i>radius</i>	Radius of the rose
<i>north</i>	Position pointing to north
<i>width</i>	Width of the rose
<i>numThorns</i>	Number of thorns
<i>numThorn-Levels</i>	Number of thorn levels
<i>shrinkFactor</i>	Factor to shrink the thorns with each level

12.78.3.3 int QwtSimpleCompassRose::numThornLevels () const

Returns

Number of thorn levels

See also

[setNumThorns\(\)](#), [setNumThornLevels\(\)](#)

12.78.3.4 int QwtSimpleCompassRose::numThorns () const

Returns

Number of thorns

See also

[setNumThorns\(\)](#), [setNumThornLevels\(\)](#)

12.78.3.5 const QPalette& QwtCompassRose::palette () const [inline, inherited]

Returns

Current palette

12.78.3.6 void QwtSimpleCompassRose::setNumThornLevels (int *numThornLevels*)

Set the of thorns levels

Parameters

<i>numThornLevels</i>	Number of thorns levels
-----------------------	-------------------------

See also

[setNumThorns\(\)](#), [numThornLevels\(\)](#)

12.78.3.7 void QwtSimpleCompassRose::setNumThorns (int *numThorns*)

Set the number of thorns on one level The number is aligned to a multiple of 4, with a minimum of 4

Parameters

<i>numThorns</i>	Number of thorns
------------------	------------------

See also

[numThorns\(\)](#), [setNumThornLevels\(\)](#)

12.78.3.8 virtual void QwtCompassRose::setPalette (const QPalette & *p*)
[inline, virtual, inherited]

Assign a palette.

12.78.3.9 void QwtSimpleCompassRose::setWidth (double *width*)

Set the width of the rose heads. Lower value make thinner heads. The range is limited from 0.03 to 0.4.

Parameters

<i>width</i>	Width
--------------	-------

12.78.3.10 double QwtSimpleCompassRose::width () const [inline]

See also

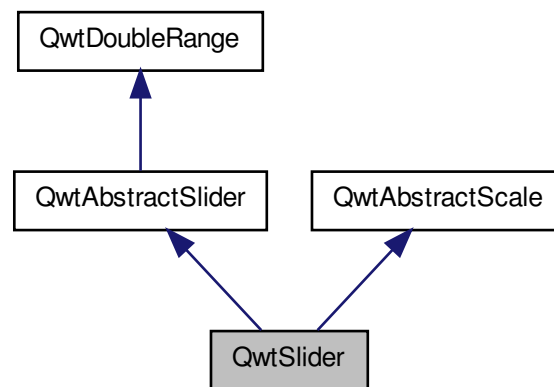
[setWidth\(\)](#)

12.79 QwtSlider Class Reference

The Slider Widget.

```
#include <qwt_slider.h>
```

Inheritance diagram for QwtSlider:



Public Types

- enum [BGSTYLE](#) {
 BgTrough = 0x1,
 BgSlot = 0x2,
 BgBoth = BgTrough | BgSlot }
- enum [ScalePos](#) {
 NoScale,
 LeftScale,
 RightScale,
 TopScale,

BottomScale }

- enum [ScrollMode](#) {
ScrNone,
ScrMouse,
ScrTimer,
ScrDirect,
ScrPage }

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- virtual void [setReadOnly](#) (bool)
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()
- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- bool [autoScale](#) () const
- [BGSTYLE](#) [bgStyle](#) () const
- int [borderWidth](#) () const
- virtual void [incPages](#) (int)
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- virtual double [mass](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const
- double [minValue](#) () const
- Qt::Orientation [orientation](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtSlider](#) (QWidget *parent, Qt::Orientation=Qt::Horizontal, [ScalePos](#)=NoScale, [BGSTYLE](#) bgStyle=BgTrough)
- const [QwtScaleDraw](#) * [scaleDraw](#) () const
- const [QwtScaleEngine](#) * [scaleEngine](#) () const
- [QwtScaleEngine](#) * [scaleEngine](#) ()
- const [QwtScaleMap](#) & [scaleMap](#) () const
- int [scaleMaxMajor](#) () const

- int [scaleMaxMinor](#) () const
- [ScalePos](#) [scalePosition](#) () const
- void [setAutoScale](#) ()
- void [setBgStyle](#) (BGSTYLE)
- void [setBorderWidth](#) (int bw)
- void [setMargins](#) (int x, int y)
- virtual void [setMass](#) (double val)
- virtual void [setOrientation](#) (Qt::Orientation)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setScale](#) (double vmin, double vmax, double step=0.0)
- void [setScale](#) (const [QwtScaleDiv](#) &s)
- void [setScale](#) (const [QwtDoubleInterval](#) &, double step=0.0)
- void [setScaleDraw](#) ([QwtScaleDraw](#) *)
- void [setScaleEngine](#) ([QwtScaleEngine](#) *)
- void [setScaleMaxMajor](#) (int ticks)
- void [setScaleMaxMinor](#) (int ticks)
- void [setScalePosition](#) ([ScalePos](#) s)
- void [setStep](#) (double)
- void [setThumbLength](#) (int l)
- void [setThumbWidth](#) (int w)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)
- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- void [stopMoving](#) ()
- int [thumbLength](#) () const
- int [thumbWidth](#) () const
- double [value](#) () const

Protected Member Functions

- const [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) () const
- [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) ()
- void [draw](#) (QPainter *p, const QRect &update_rect)
- virtual void [drawSlider](#) (QPainter *p, const QRect &r)
- virtual void [drawThumb](#) (QPainter *p, const QRect &, int pos)
- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [fontChange](#) (const QFont &oldFont)
- virtual void [getScrollMode](#) (const QPoint &p, int &scrollMode, int &direction)
- virtual double [getValue](#) (const QPoint &p)
- virtual void [keyPressEvent](#) (QKeyEvent *e)
- void [layoutSlider](#) (bool update=true)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)

- double **mouseOffset** () const
- virtual void **mousePressEvent** (QMouseEvent *e)
- virtual void **mouseReleaseEvent** (QMouseEvent *e)
- virtual void **paintEvent** (QPaintEvent *e)
- double **prevValue** () const
- virtual void **rangeChange** ()
- void **rescale** (double vmin, double vmax, double step=0.0)
- virtual void **resizeEvent** (QResizeEvent *e)
- virtual void **scaleChange** ()
- **QwtScaleDraw** * **scaleDraw** ()
- int **scrollMode** () const
- void **setAbstractScaleDraw** (QwtAbstractScaleDraw *)
- void **setMouseOffset** (double)
- virtual void **setPosition** (const QPoint &)
- virtual void **stepChange** ()
- virtual void **timerEvent** (QTimerEvent *e)
- virtual void **valueChange** ()
- virtual void **wheelEvent** (QWheelEvent *e)
- int **xyPosition** (double v) const

12.79.1 Detailed Description

The Slider Widget. [QwtSlider](#) is a slider widget which operates on an interval of type double. [QwtSlider](#) supports different layouts as well as a scale.

See also

[QwtAbstractSlider](#) and [QwtAbstractScale](#) for the descriptions of the inherited members.

12.79.2 Member Enumeration Documentation

12.79.2.1 enum QwtSlider::BGSTYLE

Background style.

See also

[QwtSlider\(\)](#)

12.79.2.2 enum QwtSlider::ScalePos

Scale position. [QwtSlider](#) tries to enforce valid combinations of its orientation and scale position:

- Qt::Horizontal combines with NoScale, TopScale and BottomScale

- Qt::Vertical combines with NoScale, LeftScale and RightScale

See also

[QwtSlider\(\)](#)

12.79.2.3 enum QwtAbstractSlider::ScrollMode [inherited]

Scroll mode

See also

[getScrollMode\(\)](#)

12.79.3 Constructor & Destructor Documentation

12.79.3.1 QwtSlider::QwtSlider (QWidget * *parent*, Qt::Orientation *orientation* = Qt::Horizontal, ScalePos *scalePos* = NoScale, BGSTYLE *bgStyle* = BgTrough) [explicit]

Constructor.

Parameters

<i>parent</i>	parent widget
<i>orientation</i>	Orientation of the slider. Can be Qt::Horizontal or Qt::Vertical. Defaults to Qt::Horizontal.
<i>scalePos</i>	Position of the scale. Defaults to QwtSlider::NoScale.
<i>bgStyle</i>	Background style. QwtSlider::BgTrough draws the slider button in a trough, QwtSlider::BgSlot draws a slot underneath the button. An or-combination of both may also be used. The default is QwtSlider::BgTrough.

[QwtSlider](#) enforces valid combinations of its orientation and scale position. If the combination is invalid, the scale position will be set to NoScale. Valid combinations are:

- Qt::Horizontal with NoScale, TopScale, or BottomScale;
- Qt::Vertical with NoScale, LeftScale, or RightScale.

12.79.4 Member Function Documentation

12.79.4.1 const QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () const [protected, inherited]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.79.4.2 QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw ()
[protected, inherited]**Returns**

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.79.4.3 bool QwtAbstractScale::autoScale () const [inherited]**Returns**

true if autoscaling is enabled

12.79.4.4 QwtSlider::BGSTYLE QwtSlider::bgStyle () const**Returns**

the background style.

12.79.4.5 int QwtSlider::borderWidth () const**Returns**

the border width.

12.79.4.6 void QwtSlider::draw (QPainter *p, const QRect &update_rect)
[protected]

Draw the [QwtSlider](#).

12.79.4.7 void QwtSlider::drawSlider (QPainter * *painter*, const QRect & *r*)
[protected, virtual]

Draw the slider into the specified rectangle.

Parameters

<i>painter</i>	Painter
<i>r</i>	Rectangle

12.79.4.8 void QwtSlider::drawThumb (QPainter * *painter*, const QRect & *sliderRect*, int *pos*) [protected, virtual]

Draw the thumb at a position

Parameters

<i>painter</i>	Painter
<i>sliderRect</i>	Bounding rectangle of the slider
<i>pos</i>	Position of the slider thumb

12.79.4.9 double QwtDoubleRange::exactPrevValue () const [protected, inherited]

Returns the exact previous value.

12.79.4.10 double QwtDoubleRange::exactValue () const [protected, inherited]

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.79.4.11 void QwtAbstractSlider::fitValue (double *value*) [virtual, slot, inherited]

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.79.4.12 `void QwtSlider::fontChange (const QFont & oldFont)`
[protected, virtual]

Notify change in font.

12.79.4.13 `void QwtSlider::getScrollMode (const QPoint & p, int & scrollMode, int & direction)` [protected, virtual]

Determine scrolling mode and direction.

Parameters

<i>p</i>	point
<i>scrollMode</i>	Scrolling mode
<i>direction</i>	Direction

Implements [QwtAbstractSlider](#).

12.79.4.14 `double QwtSlider::getValue (const QPoint & pos)` [protected, virtual]

Determine the value corresponding to a specified mouse location.

Parameters

<i>pos</i>	Mouse position
------------	----------------

Implements [QwtAbstractSlider](#).

12.79.4.15 `void QwtDoubleRange::incPages (int nPages)` [virtual, inherited]

Increment the value by a specified number of pages.

Parameters

<i>nPages</i> Number of pages to increment. A negative number decrements the value.

Warning

The Page size is specified in the constructor.

12.79.4.16 `void QwtAbstractSlider::incValue (int steps) [virtual, slot, inherited]`

Increment the value by a specified number of steps.

Parameters

<i>steps</i> number of steps

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.79.4.17 `bool QwtAbstractSlider::isReadOnly () const [inherited]`

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.79.4.18 `bool QwtAbstractSlider::isValid () const [inline, inherited]`

See also

[QwtDbfRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.79.4.19 `void QwtAbstractSlider::keyPressEvent (QKeyEvent * e)`
[protected, virtual, inherited]

Handles key events

- Key_Down, Key_Left
Decrement by 1
- Key_Up, Key_Right
Increment by 1

Parameters

<i>e</i>	Key event
----------	-----------

See also

[isReadOnly\(\)](#)

Reimplemented in [QwtCompass](#), and [QwtDial](#).

12.79.4.20 `void QwtSlider::layoutSlider (bool update_geometry = true)`
[protected]

Recalculate the slider's geometry and layout based on the current rect and fonts.

Parameters

<i>update_geometry</i>	notify the layout system and call update to redraw the scale
------------------------	--

12.79.4.21 `double QwtAbstractSlider::mass () const` **[virtual, inherited]**

Returns

mass

See also

[setMass\(\)](#)

Reimplemented in [QwtWheel](#).

12.79.4.22 `double QwtDoubleRange::maxValue () const` **[inherited]**

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.79.4.23 QSize QwtSlider::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

The return value of [QwtSlider::minimumSizeHint\(\)](#) depends on the font and the scale.

12.79.4.24 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

12.79.4.25 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * e) [protected, virtual, inherited]

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.79.4.26 void QwtAbstractSlider::mousePressEvent (QMouseEvent * e) [protected, virtual, inherited]

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.79.4.27 `void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.79.4.28 `Qt::Orientation QwtAbstractSlider::orientation () const`
[inherited]

Returns

Orientation

See also

[setOrientation\(\)](#)

12.79.4.29 `int QwtDoubleRange::pageSize () const` [inherited]

Returns the page size in steps.

12.79.4.30 `void QwtSlider::paintEvent (QPaintEvent * event)`
[protected, virtual]

Qt paint event

Parameters

<i>event</i>	Paint event
--------------	-------------

12.79.4.31 `bool QwtDoubleRange::periodic () const` [inherited]

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.79.4.32 `double QwtDoubleRange::prevValue () const` `[protected, inherited]`

Returns the previous value.

12.79.4.33 `void QwtSlider::rangeChange ()` `[protected, virtual]`

Notify change of range.

Reimplemented from [QwtDoubleRange](#).

12.79.4.34 `void QwtAbstractScale::rescale (double vmin, double vmax, double stepSize = 0.0)` `[protected, inherited]`

Recalculate the scale division and update the scale draw.

Parameters

<i>vmin</i>	Lower limit of the scale interval
<i>vmax</i>	Upper limit of the scale interval
<i>stepSize</i>	Major step size

See also

[scaleChange\(\)](#)

12.79.4.35 `void QwtSlider::resizeEvent (QResizeEvent * e)` `[protected, virtual]`

Qt resize event.

12.79.4.36 `void QwtSlider::scaleChange ()` `[protected, virtual]`

Notify changed scale.

Reimplemented from [QwtAbstractScale](#).

12.79.4.37 `const QwtScaleDraw * QwtSlider::scaleDraw () const`

Returns

the scale draw of the slider

See also

[setScaleDraw\(\)](#)

12.79.4.38 QwtScaleDraw * QwtSlider::scaleDraw () [protected]**Returns**

the scale draw of the slider

See also

[setScaleDraw\(\)](#)

12.79.4.39 const QwtScaleEngine * QwtAbstractScale::scaleEngine () const [inherited]**Returns**

Scale engine

See also

[setScaleEngine\(\)](#)

12.79.4.40 QwtScaleEngine * QwtAbstractScale::scaleEngine () [inherited]**Returns**

Scale engine

See also

[setScaleEngine\(\)](#)

12.79.4.41 `const QwtScaleMap & QwtAbstractScale::scaleMap () const` `[inherited]`

Returns

`abstractScaleDraw()->scaleMap()`

12.79.4.42 `int QwtAbstractScale::scaleMaxMajor () const` `[inherited]`

Returns

Max. number of major tick intervals The default value is 5.

12.79.4.43 `int QwtAbstractScale::scaleMaxMinor () const` `[inherited]`

Returns

Max. number of minor tick intervals The default value is 3.

12.79.4.44 `QwtSlider::ScalePos QwtSlider::scalePosition () const`

Return the scale position.

12.79.4.45 `void QwtAbstractScale::setAbstractScaleDraw (`
`QwtAbstractScaleDraw * scaleDraw)` `[protected,`
`inherited]`

Set a scale draw.

`scaleDraw` has to be created with `new` and will be deleted in `~QwtAbstractScale` or the next call of `setAbstractScaleDraw`.

12.79.4.46 `void QwtAbstractScale::setAutoScale ()` `[inherited]`

Advise the widget to control the scale range internally.

Autoscaling is on by default.

See also

[setScale\(\)](#), [autoScale\(\)](#)

12.79.4.47 void QwtSlider::setBgStyle (BGSTYLE *st*)

Set the background style.

12.79.4.48 void QwtSlider::setBorderWidth (int *bd*)

Change the slider's border width.

Parameters

<i>bd</i>	border width
-----------	--------------

12.79.4.49 void QwtSlider::setMargins (int *xMargin*, int *yMargin*)

Set distances between the widget's border and internals.

Parameters

<i>xMargin</i>	Horizontal margin
<i>yMargin</i>	Vertical margin

12.79.4.50 void QwtAbstractSlider::setMass (double *val*) [virtual, inherited]

Set the slider's mass for flywheel effect.

If the slider's mass is greater than 0, it will continue to move after the mouse button has been released. Its speed decreases with time at a rate depending on the slider's mass. A large mass means that it will continue to move for a long time.

Derived widgets may overload this function to make it public.

Parameters

<i>val</i>	New mass in kg
------------	----------------

See also

[mass\(\)](#)

Reimplemented in [QwtWheel](#).

12.79.4.51 void QwtSlider::setOrientation (Qt::Orientation *o*) [virtual]

Set the orientation.

Parameters

<i>o</i>	Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.
----------	--

If the new orientation and the old scale position are an invalid combination, the scale position will be set to QwtSlider::NoScale.

See also

[QwtAbstractSlider::orientation\(\)](#)

Reimplemented from [QwtAbstractSlider](#).

12.79.4.52 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.79.4.53 void QwtAbstractSlider::setPosition (const QPoint & *p*) [protected, virtual, inherited]

Move the slider to a specified point, adjust the value and emit signals if necessary.

12.79.4.54 void QwtDoubleRange::setRange (double *vmin*, double *vmax*, double *vstep* = 0.0, int *pageSize* = 1) [inherited]

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will *not* be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.79.4.55 `void QwtAbstractSlider::setReadOnly (bool readOnly)`
[virtual, slot, inherited]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also

[isReadOnly\(\)](#)

12.79.4.56 `void QwtAbstractScale::setScale (const QwtDoubleInterval & interval, double stepSize = 0.0)` **[inherited]**

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>interval</i>	Interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.79.4.57 void QwtAbstractScale::setScale (double *vmin*, double *vmax*, double *stepSize* = 0.0) [inherited]

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>vmin</i>	lower limit of the scale interval
<i>vmax</i>	upper limit of the scale interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.79.4.58 void QwtAbstractScale::setScale (const QwtScaleDiv & *scaleDiv*) [inherited]

Specify a scale.

Disable autoscaling and define a scale by a scale division

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

See also

[setAutoScale\(\)](#)

12.79.4.59 void QwtSlider::setScaleDraw (QwtScaleDraw * *scaleDraw*)

Set a scale draw.

For changing the labels of the scales, it is necessary to derive from [QwtScaleDraw](#) and overload [QwtScaleDraw::label\(\)](#).

Parameters

<i>scaleDraw</i>	ScaleDraw object, that has to be created with new and will be deleted in ~QwtSlider or the next call of setScaleDraw() .
------------------	--

12.79.4.60 void QwtAbstractScale::setScaleEngine (QwtScaleEngine *
scaleEngine) [inherited]

Set a scale engine.

The scale engine is responsible for calculating the scale division, and in case of auto scaling how to align the scale.

scaleEngine has to be created with new and will be deleted in ~QwtAbstractScale or the next call of setScaleEngine.

12.79.4.61 void QwtAbstractScale::setScaleMaxMajor (int *ticks*)
[inherited]

Set the maximum number of major tick intervals.

The scale's major ticks are calculated automatically such that the number of major intervals does not exceed ticks. The default value is 5.

Parameters

<i>ticks</i> maximal number of major ticks.

See also

[QwtAbstractScaleDraw](#)

12.79.4.62 void QwtAbstractScale::setScaleMaxMinor (int *ticks*)
[inherited]

Set the maximum number of minor tick intervals.

The scale's minor ticks are calculated automatically such that the number of minor intervals does not exceed ticks. The default value is 3.

Parameters

<i>ticks</i>

See also

[QwtAbstractScaleDraw](#)

12.79.4.63 void QwtSlider::setScalePosition (ScalePos *s*)

Change the scale position (and slider orientation).

Parameters

<i>s</i>	Position of the scale.
----------	------------------------

A valid combination of scale position and orientation is enforced:

- if the new scale position is Left or Right, the scale orientation will become Qt::Vertical;
- if the new scale position is Bottom or Top the scale orientation will become Qt::Horizontal;
- if the new scale position is QwtSlider::NoScale, the scale orientation will not change.

12.79.4.64 void QwtDoubleRange::setStep (double *vstep*) [inherited]

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

12.79.4.65 void QwtSlider::setThumbLength (int *thumbLength*)

Set the slider's thumb length.

Parameters

<i>thumbLength</i>	new length
--------------------	------------

12.79.4.66 void QwtSlider::setThumbWidth (int *w*)

Change the width of the thumb.

Parameters

<i>w</i>	new width
----------	-----------

**12.79.4.67 void QwtAbstractSlider::setTracking (bool *enable*)
[inherited]**

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i>	true (enable) or false (disable) tracking.
---------------	--

12.79.4.68 void QwtAbstractSlider::setUpdateTime (int *t*) [inherited]

Specify the update interval for automatic scrolling.

Parameters

<i>t</i>	update interval in milliseconds
----------	---------------------------------

See also

[getScrollMode\(\)](#)

12.79.4.69 void QwtAbstractSlider::setValid (bool *valid*) [inline, inherited]**Parameters**

<i>valid</i>	true/false
--------------	------------

See also

[QwtDbfRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.79.4.70 `void QwtAbstractSlider::setValue (double val) [virtual, slot, inherited]`

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i> new value

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.79.4.71 `QSize QwtSlider::sizeHint () const [virtual]`

Returns

[QwtSlider::minimumSizeHint\(\)](#)

12.79.4.72 `void QwtAbstractSlider::sliderMoved (double value) [signal, inherited]`

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i> new value

12.79.4.73 `void QwtAbstractSlider::sliderPressed () [signal, inherited]`

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.79.4.74 void QwtAbstractSlider::sliderReleased () [signal, inherited]

This signal is emitted when the user releases the movable part of the slider.

12.79.4.75 double QwtDoubleRange::step () const [inherited]

Returns

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.79.4.76 void QwtDoubleRange::stepChange () [protected, virtual, inherited]

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.79.4.77 void QwtAbstractSlider::stopMoving () [inherited]

Stop updating if automatic scrolling is active.

12.79.4.78 int QwtSlider::thumbLength () const

Returns

the thumb length.

12.79.4.79 int QwtSlider::thumbWidth () const

Returns

the thumb width.

12.79.4.80 void QwtAbstractSlider::timerEvent (QTimerEvent * *e*)
[protected, virtual, inherited]

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.79.4.81 double QwtDoubleRange::value () const [inherited]

Returns the current value.

Reimplemented in [QwtCounter](#).

12.79.4.82 void QwtSlider::valueChange () [protected, virtual]

Notify change of value.

Reimplemented from [QwtAbstractSlider](#).

12.79.4.83 void QwtAbstractSlider::valueChanged (double *value*)
[signal, inherited]

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.79.4.84 void QwtAbstractSlider::wheelEvent (QWheelEvent * *e*)
[protected, virtual, inherited]

Wheel Event handler

Parameters

<i>e</i>	Wheel event
----------	-------------

12.79.4.85 int QwtSlider::xyPosition (double *value*) const [protected]

Find the x/y position for a given value v

Parameters

<i>value</i>	Value
--------------	-------

12.80 QwtSpline Class Reference

A class for spline interpolation.

#include <qwt_spline.h>

Public Types

- enum [SplineType](#) {
 Natural,
 Periodic }

Public Member Functions

- const QwtArray< double > & [coefficientsA](#) () const
- const QwtArray< double > & [coefficientsB](#) () const
- const QwtArray< double > & [coefficientsC](#) () const
- bool [isValid](#) () const
- [QwtSpline](#) & [operator=](#) (const [QwtSpline](#) &)
- QPolygonF [points](#) () const
- [QwtSpline](#) ()
- [QwtSpline](#) (const [QwtSpline](#) &)
- void [reset](#) ()
- bool [setPoints](#) (const QPolygonF &points)
- void [setSplineType](#) ([SplineType](#))
- [SplineType](#) [splineType](#) () const
- double [value](#) (double x) const
- [~QwtSpline](#) ()

Protected Member Functions

- bool [buildNaturalSpline](#) (const QPolygonF &)
- bool [buildPeriodicSpline](#) (const QPolygonF &)

Protected Attributes

- PrivateData * **d_data**

12.80.1 Detailed Description

A class for spline interpolation. The [QwtSpline](#) class is used for cubical spline interpolation. Two types of splines, natural and periodic, are supported.

Usage:

1. First call [setPoints\(\)](#) to determine the spline coefficients for a tabulated function $y(x)$.
2. After the coefficients have been set up, the interpolated function value for an argument x can be determined by calling [QwtSpline::value\(\)](#).

Example:

```
#include <qwt_spline.h>

QPolygonF interpolate(const QPolygonF& points, int numValues)
{
    QwtSpline spline;
    if ( !spline.setPoints(points) )
        return points;

    QPolygonF interpolatedPoints(numValues);

    const double delta =
        (points[numPoints - 1].x() - points[0].x()) / (points.size() - 1);
    for(i = 0; i < points.size(); i++) / interpolate
    {
        const double x = points[0].x() + i * delta;
        interpolatedPoints[i].setX(x);
        interpolatedPoints[i].setY(spline.value(x));
    }
    return interpolatedPoints;
}
```

12.80.2 Member Enumeration Documentation

12.80.2.1 enum QwtSpline::SplineType

Spline type.

12.80.3 Constructor & Destructor Documentation

12.80.3.1 QwtSpline::QwtSpline ()

Constructor.

12.80.3.2 QwtSpline::QwtSpline (const QwtSpline & *other*)

Copy constructor

Parameters

<i>other</i>	Spline used for initialization
--------------	--------------------------------

12.80.3.3 QwtSpline::~~QwtSpline ()

Destructor.

12.80.4 Member Function Documentation**12.80.4.1 bool QwtSpline::buildNaturalSpline (const QPolygonF & *points*)
[protected]**

Determines the coefficients for a natural spline.

Returns

true if successful

**12.80.4.2 bool QwtSpline::buildPeriodicSpline (const QPolygonF & *points*)
[protected]**

Determines the coefficients for a periodic spline.

Returns

true if successful

12.80.4.3 const QwtArray< double > & QwtSpline::coefficientsA () const**Returns**

A coefficients

12.80.4.4 `const QwtArray< double > & QwtSpline::coefficientsB () const`**Returns**

B coefficients

12.80.4.5 `const QwtArray< double > & QwtSpline::coefficientsC () const`**Returns**

C coefficients

12.80.4.6 `bool QwtSpline::isValid () const`

True if valid.

12.80.4.7 `QwtSpline & QwtSpline::operator= (const QwtSpline & other)`

Assignment operator

Parameters

<i>other</i>	Spline used for initialization
--------------	--------------------------------

12.80.4.8 `QPolygonF QwtSpline::points () const`

Return points passed by setPoints

12.80.4.9 `void QwtSpline::reset ()`

Free allocated memory and set size to 0.

12.80.4.10 `bool QwtSpline::setPoints (const QPolygonF & points)`

Calculate the spline coefficients.

Depending on the value of *periodic*, this function will determine the coefficients for a natural or a periodic spline and store them internally.

Parameters

<i>points</i>	Points
---------------	--------

Returns

true if successful

Warning

The sequence of x (but not y) values has to be strictly monotone increasing, which means `points[i].x() < points[i+1].x()`. If this is not the case, the function will return false

12.80.4.11 void QwtSpline::setSplineType (SplineType *splineType*)

Select the algorithm used for calculating the spline

Parameters

<i>splineType</i>	Spline type
-------------------	-------------

See also

[splineType\(\)](#)

12.80.4.12 QwtSpline::SplineType QwtSpline::splineType () const**Returns**

the spline type

See also

[setSplineType\(\)](#)

12.80.4.13 double QwtSpline::value (double *x*) const

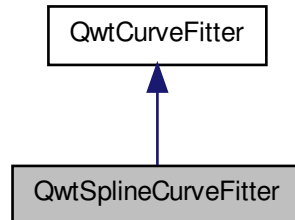
Calculate the interpolated function value corresponding to a given argument x.

12.81 QwtSplineCurveFitter Class Reference

A curve fitter using cubic splines.

```
#include <qwt_curve_fitter.h>
```


Inheritance diagram for QwtSplineCurveFitter:



Public Types

- enum **FitMode** {
 Auto,
 Spline,
 ParametricSpline }

Public Member Functions

- virtual QPolygonF **fitCurve** (const QPolygonF &) const
- FitMode **fitMode** () const
- **QwtSplineCurveFitter** ()
- void **setFitMode** (FitMode)
- void **setSpline** (const **QwtSpline** &)
- void **setSplineSize** (int size)
- **QwtSpline** & **spline** ()
- const **QwtSpline** & **spline** () const
- int **splineSize** () const
- virtual **~QwtSplineCurveFitter** ()

12.81.1 Detailed Description

A curve fitter using cubic splines.

12.81.2 Constructor & Destructor Documentation

12.81.2.1 QwtSplineCurveFitter::QwtSplineCurveFitter ()

Constructor.

12.81.2.2 QwtSplineCurveFitter::~~QwtSplineCurveFitter () [virtual]

Destructor.

12.81.3 Member Function Documentation

12.81.3.1 QPolygonF QwtSplineCurveFitter::fitCurve (const QPolygonF & *points*) const [virtual]

Find a curve which has the best fit to a series of data points

Parameters

<i>points</i>	Series of data points
---------------	-----------------------

Returns

Curve points

Implements [QwtCurveFitter](#).

12.81.3.2 QwtSplineCurveFitter::FitMode QwtSplineCurveFitter::fitMode () const

Returns

Mode representing a spline algorithm

See also

[setFitMode\(\)](#)

12.81.3.3 void QwtSplineCurveFitter::setFitMode (FitMode *mode*)

Select the algorithm used for building the spline

Parameters

<i>mode</i>	Mode representing a spline algorithm
-------------	--------------------------------------

See also

[fitMode\(\)](#)

12.81.3.4 void QwtSplineCurveFitter::setSplineSize (int *splineSize*)

Assign a spline size (has to be at least 10 points)

Parameters

<i>splineSize</i>	Spline size
-------------------	-------------

See also

[splineSize\(\)](#)

12.81.3.5 int QwtSplineCurveFitter::splineSize () const

Returns

Spline size

See also

[setSplineSize\(\)](#)

12.82 QwtSymbol Class Reference

A class for drawing symbols.

```
#include <qwt_symbol.h>
```

Public Types

- enum [Style](#) {
NoSymbol = -1,
Ellipse,
Rect,
Diamond,
Triangle,
DTriangle,

UTriangle,
LTriangle,
RTriangle,
Cross,
XCross,
HLine,
VLine,
Star1,
Star2,
Hexagon,
StyleCnt }

Public Member Functions

- const QBrush & [brush](#) () const
- virtual [QwtSymbol](#) * [clone](#) () const
- void [draw](#) (QPainter *p, const QPoint &pt) const
- void [draw](#) (QPainter *p, int x, int y) const
- virtual void [draw](#) (QPainter *p, const QRect &r) const
- bool [operator!=](#) (const [QwtSymbol](#) &) const
- virtual bool [operator==](#) (const [QwtSymbol](#) &) const
- const QPen & [pen](#) () const
- [QwtSymbol](#) ([Style](#) st, const QBrush &bd, const QPen &pn, const QSize &s)
- [QwtSymbol](#) ()
- void [setBrush](#) (const QBrush &b)
- void [setPen](#) (const QPen &p)
- void [setSize](#) (int a, int b=-1)
- void [setSize](#) (const QSize &s)
- void [setStyle](#) ([Style](#) s)
- const QSize & [size](#) () const
- [Style](#) [style](#) () const
- virtual [~QwtSymbol](#) ()

12.82.1 Detailed Description

A class for drawing symbols.

12.82.2 Member Enumeration Documentation

12.82.2.1 enum [QwtSymbol::Style](#)

[Style](#)

See also

[setStyle\(\)](#), [style\(\)](#)

12.82.3 Constructor & Destructor Documentation

12.82.3.1 QwtSymbol::QwtSymbol ()

Default Constructor

The symbol is constructed with gray interior, black outline with zero width, no size and style 'NoSymbol'.

12.82.3.2 QwtSymbol::QwtSymbol (QwtSymbol::Style *style*, const QBrush & *brush*, const QPen & *pen*, const QSize & *size*)

Constructor.

Parameters

<i>style</i>	Symbol Style
<i>brush</i>	brush to fill the interior
<i>pen</i>	outline pen
<i>size</i>	size

12.82.3.3 QwtSymbol::~~QwtSymbol () [virtual]

Destructor.

12.82.4 Member Function Documentation

12.82.4.1 const QBrush& QwtSymbol::brush () const [inline]

Return Brush.

12.82.4.2 QwtSymbol * QwtSymbol::clone () const [virtual]

Allocate and return a symbol with the same attributes

Returns

Cloned symbol

12.82.4.3 void QwtSymbol::draw (QPainter * *painter*, const QRect & *r*) const [virtual]

Draw the symbol into a bounding rectangle.

This function assumes that the painter has been initialized with brush and pen before. This allows a much more performant implementation when painting many symbols with the same brush and pen like in curves.

Parameters

<i>painter</i>	Painter
<i>r</i>	Bounding rectangle

12.82.4.4 void QwtSymbol::draw (QPainter **painter*, const QPoint &*pos*) const

Draw the symbol at a specified point.

Parameters

<i>painter</i>	Painter
<i>pos</i>	Center of the symbol

12.82.4.5 void QwtSymbol::draw (QPainter **p*, int *x*, int *y*) const

Draw the symbol at a point (x,y).

12.82.4.6 bool QwtSymbol::operator!= (const QwtSymbol &*other*) const

!= operator

12.82.4.7 bool QwtSymbol::operator== (const QwtSymbol &*other*) const
[virtual]

== operator

12.82.4.8 const QPen& QwtSymbol::pen () const [inline]

Return Pen.

12.82.4.9 void QwtSymbol::setBrush (const QBrush & *brush*)

Assign a brush.

The brush is used to draw the interior of the symbol.

Parameters

<i>brush</i>	Brush
--------------	-------

12.82.4.10 void QwtSymbol::setPen (const QPen & *pen*)

Assign a pen

The pen is used to draw the symbol's outline.

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

<i>pen</i>	Pen
------------	-----

See also

[pen\(\)](#), [setBrush\(\)](#), [QwtPainter::scaledPen\(\)](#)

12.82.4.11 void QwtSymbol::setSize (const QSize & *size*)

Set the symbol's size

Parameters

<i>size</i>	Size
-------------	------

12.82.4.12 void QwtSymbol::setSize (int *width*, int *height* = -1)

Specify the symbol's size.

If the 'h' parameter is left out or less than 0, and the 'w' parameter is greater than or equal to 0, the symbol size will be set to (w,w).

Parameters

<i>width</i>	Width
<i>height</i>	Height (defaults to -1)

12.82.4.13 void QwtSymbol::setStyle (QwtSymbol::Style s)

Specify the symbol style.

The following styles are defined:

NoSymbol No Style. The symbol cannot be drawn.

Ellipse Ellipse or circle

Rect Rectangle

Diamond Diamond

Triangle Triangle pointing upwards

DTriangle Triangle pointing downwards

UTriangle Triangle pointing upwards

LTriangle Triangle pointing left

RTriangle Triangle pointing right

Cross Cross (+)

XCross Diagonal cross (X)

HLine Horizontal line

VLine Vertical line

Star1 X combined with +

Star2 Six-pointed star

Hexagon Hexagon

Parameters

<i>s</i>	style
----------	-------

12.82.4.14 const QSize& QwtSymbol::size () const [inline]

Return Size.

12.82.4.15 Style QwtSymbol::style () const [inline]

Return Style.

12.83 QwtText Class Reference

A class representing a text.

```
#include <qwt_text.h>
```

Public Types

- enum [LayoutAttribute](#) { **MinimumLayout** = 1 }
- enum [PaintAttribute](#) {
 PaintUsingTextFont = 1,
 PaintUsingTextColor = 2,
 PaintBackground = 4 }
- enum [TextFormat](#) {
 AutoText = 0,
 PlainText,
 RichText,
 MathMLText,
 TeXText,
 OtherFormat = 100 }

Public Member Functions

- [QBrush](#) [backgroundBrush](#) () const
- [QPen](#) [backgroundPen](#) () const
- [QColor](#) [color](#) () const
- void [draw](#) ([QPainter](#) *painter, const [QRect](#) &rect) const
- [QFont](#) [font](#) () const
- int [heightForWidth](#) (int width, const [QFont](#) &=QFont()) const
- bool [isEmpty](#) () const
- bool [isNull](#) () const
- int [operator!=](#) (const [QwtText](#) &) const
- [QwtText](#) & [operator=](#) (const [QwtText](#) &)
- int [operator==](#) (const [QwtText](#) &) const
- [QwtText](#) (const [QwtText](#) &)
- [QwtText](#) (const [QString](#) &=QString::null, [TextFormat](#) textFormat=AutoText)
- int [renderFlags](#) () const
- void [setBackgroundBrush](#) (const [QBrush](#) &)
- void [setBackgroundPen](#) (const [QPen](#) &)
- void [setColor](#) (const [QColor](#) &)
- void [setFont](#) (const [QFont](#) &)
- void [setLayoutAttribute](#) ([LayoutAttribute](#), bool on=true)
- void [setPaintAttribute](#) ([PaintAttribute](#), bool on=true)
- void [setRenderFlags](#) (int flags)

- void [setText](#) (const QString &, [QwtText::TextFormat](#) textFormat=AutoText)
- bool [testLayoutAttribute](#) ([LayoutAttribute](#)) const
- bool [testPaintAttribute](#) ([PaintAttribute](#)) const
- QString [text](#) () const
- QSize [textSize](#) (const QFont &=QFont()) const
- QColor [usedColor](#) (const QColor &) const
- QFont [usedFont](#) (const QFont &) const
- [~QwtText](#) ()

Static Public Member Functions

- static void [setTextEngine](#) ([QwtText::TextFormat](#), [QwtTextEngine](#) *)
- static const [QwtTextEngine](#) * [textEngine](#) (const QString &text, [QwtText::TextFormat](#)=AutoText)
- static const [QwtTextEngine](#) * [textEngine](#) ([QwtText::TextFormat](#))

12.83.1 Detailed Description

A class representing a text. A [QwtText](#) is a text including a set of attributes how to render it.

- Format
A text might include control sequences (f.e tags) describing how to render it. Each format (f.e MathML, TeX, Qt Rich Text) has its own set of control sequences, that can be handles by a [QwtTextEngine](#) for this format.
- Background
A text might have a background, defined by a QPen and QBrush to improve its visibility.
- Font
A text might have an individual font.
- Color
A text might have an individual color.
- Render Flags
Flags from Qt::AlignmentFlag and Qt::TextFlag used like in QPainter::drawText.

See also

[QwtTextEngine](#), [QwtTextLabel](#)

12.83.2 Member Enumeration Documentation

12.83.2.1 enum QwtText::LayoutAttribute

Layout Attributes.

The layout attributes affects some aspects of the layout of the text.

- `MinimumLayout`

Layout the text without its margins. This mode is useful if a text needs to be aligned accurately, like the tick labels of a scale. If [QwtTextEngine::textMargins](#) is not implemented for the format of the text, `MinimumLayout` has no effect.

12.83.2.2 enum QwtText::PaintAttribute

Paint Attributes.

Font and color and background are optional attributes of a [QwtText](#). The paint attributes hold the information, if they are set.

- `PaintUsingTextFont`

The text has an individual font.

- `PaintUsingTextColor`

The text has an individual color.

- `PaintBackground`

The text has an individual background.

12.83.2.3 enum QwtText::TextFormat

Text format.

The text format defines the [QwtTextEngine](#), that is used to render the text.

- `AutoText`

The text format is determined using [QwtTextEngine::mightRender](#) for all available text engines in increasing order > `PlainText`. If none of the text engines can render the text is rendered like `PlainText`.

- `PlainText`

Draw the text as it is, using a [QwtPlainTextEngine](#).

- **RichText**
Use the Scribe framework (Qt Rich Text) to render the text.
- **MathMLText**
Use a MathML (<http://en.wikipedia.org/wiki/MathML>) render engine to display the text. The Qwt MathML extension offers such an engine based on the MathML renderer of the Qt solutions package. Unfortunately it is only available for owners of a commercial Qt license.
- **TeXText**
Use a TeX (<http://en.wikipedia.org/wiki/TeX>) render engine to display the text.
- **OtherFormat**
The number of text formats can be extended using `setTextEngine`. Formats `>= OtherFormat` are not used by Qwt.

See also

[QwtTextEngine](#), [setTextEngine\(\)](#)

12.83.3 Constructor & Destructor Documentation
**12.83.3.1 QwtText::QwtText (const QString & *text* = *QString::null*,
QwtText::TextFormat *textFormat* = *AutoText*)**

Constructor

Parameters

<i>text</i>	Text content
<i>textFormat</i>	Text format

12.83.3.2 QwtText::QwtText (const QwtText & *other*)

Copy constructor.

12.83.3.3 QwtText::~QwtText ()

Destructor.

12.83.4 Member Function Documentation

12.83.4.1 QBrush QwtText::backgroundBrush () const

Returns

Background brush

See also

[setBackgroundBrush\(\)](#), [backgroundPen\(\)](#)

12.83.4.2 QPen QwtText::backgroundPen () const

Returns

Background pen

See also

[setBackgroundPen\(\)](#), [backgroundBrush\(\)](#)

12.83.4.3 QColor QwtText::color () const

Return the pen color, used for painting the text.

12.83.4.4 void QwtText::draw (QPainter * *painter*, const QRect & *rect*) const

Draw a text into a rectangle

Parameters

<i>painter</i>	Painter
<i>rect</i>	Rectangle

12.83.4.5 QFont QwtText::font () const

Return the font.

12.83.4.6 `int QwtText::heightForWidth (int width, const QFont & defaultFont = QFont ()) const`

Find the height for a given width

Parameters

<i>defaultFont</i>	Font, used for the calculation if the text has no font
<i>width</i>	Width

Returns

Calculated height

12.83.4.7 `bool QwtText::isEmpty () const [inline]`

Returns

`text().isEmpty()`

12.83.4.8 `bool QwtText::isNull () const [inline]`

Returns

`text().isNull()`

12.83.4.9 `int QwtText::operator!= (const QwtText & other) const`

Relational operator.

12.83.4.10 `QwtText & QwtText::operator= (const QwtText & other)`

Assignment operator.

12.83.4.11 `int QwtText::operator== (const QwtText & other) const`

Relational operator.

12.83.4.12 int QwtText::renderFlags () const**Returns**

Render flags

See also

[setRenderFlags\(\)](#)

12.83.4.13 void QwtText::setBackgroundBrush (const QBrush & *brush*)

Set the background brush

Parameters

<i>brush</i>	Background brush
--------------	------------------

See also

[backgroundBrush\(\)](#), [setBackgroundPen\(\)](#)

12.83.4.14 void QwtText::setBackgroundPen (const QPen & *pen*)

Set the background pen

Parameters

<i>pen</i>	Background pen
------------	----------------

See also

[backgroundPen\(\)](#), [setBackgroundBrush\(\)](#)

12.83.4.15 void QwtText::setColor (const QColor & *color*)

Set the pen color used for painting the text.

Parameters

<i>color</i>	Color
--------------	-------

Note

Setting the color might have no effect, when the text contains control sequences for setting colors.

12.83.4.16 void QwtText::setFont (const QFont & *font*)

Set the font.

Parameters

<i>font</i>	Font
-------------	------

Note

Setting the font might have no effect, when the text contains control sequences for setting fonts.

12.83.4.17 void QwtText::setLayoutAttribute (LayoutAttribute *attribute*, bool *on* = *true*)

Change a layout attribute

Parameters

<i>attribute</i>	Layout attribute
<i>on</i>	On/Off

See also

[testLayoutAttribute\(\)](#)

12.83.4.18 void QwtText::setPaintAttribute (PaintAttribute *attribute*, bool *on* = *true*)

Change a paint attribute

Parameters

<i>attribute</i>	Paint attribute
<i>on</i>	On/Off

Note

Used by [setFont\(\)](#), [setColor\(\)](#), [setBackgroundPen\(\)](#) and [setBackgroundBrush\(\)](#)

See also

[testPaintAttribute\(\)](#)

12.83.4.19 void QwtText::setRenderFlags (int *renderFlags*)

Change the render flags.

The default setting is Qt::AlignCenter

Parameters

<i>renderFlags</i>	Bitwise OR of the flags used like in QPainter::drawText
--------------------	---

See also

[renderFlags\(\)](#), [QwtTextEngine::draw\(\)](#)

Note

Some renderFlags might have no effect, depending on the text format.

12.83.4.20 `void QwtText::setText (const QString & text, QwtText::TextFormat textFormat = AutoText)`

Assign a new text content

Parameters

<i>text</i>	Text content
<i>textFormat</i>	Text format

See also

[text\(\)](#)

12.83.4.21 `void QwtText::setTextEngine (QwtText::TextFormat format, QwtTextEngine * engine) [static]`

Assign/Replace a text engine for a text format

With setTextEngine it is possible to extend Qwt with other types of text formats.

Owner of a commercial Qt license can build the qwtmathml library, that is based on the MathML renderer, that is included in MML Widget component of the Qt solutions package.

For QwtText::PlainText it is not allowed to assign a engine == NULL.

Parameters

<i>format</i>	Text format
<i>engine</i>	Text engine

See also

[QwtMathMLTextEngine](#)

Warning

Using QwtText::AutoText does nothing.

12.83.4.22 `bool QwtText::testLayoutAttribute (LayoutAttribute attribute) const`

Test a layout attribute

Parameters

<i>attribute</i>	Layout attribute
------------------	------------------

Returns

true, if attribute is enabled

See also

[setLayoutAttribute\(\)](#)

12.83.4.23 `bool QwtText::testPaintAttribute (PaintAttribute attribute) const`

Test a paint attribute

Parameters

<i>attribute</i>	Paint attribute
------------------	-----------------

Returns

true, if attribute is enabled

See also

[setPaintAttribute\(\)](#)

12.83.4.24 `QString QwtText::text () const`

Return the text.

See also

[setText\(\)](#)

12.83.4.25 `const QwtTextEngine * QwtText::textEngine (QwtText::TextFormat format) [static]`

Find the text engine for a text format.

textEngine can be used to find out if a text format is supported. F.e, if one wants to use MathML labels, the MathML renderer from the commercial Qt solutions package might be required, that is not available in Qt Open Source Edition environments.

Parameters

<i>format</i>	Text format
---------------	-------------

Returns

The text engine, or NULL if no engine is available.

12.83.4.26 `const QwtTextEngine * QwtText::textEngine (const QString & text, QwtText::TextFormat format = AutoText) [static]`

Find the text engine for a text format

In case of QwtText::AutoText the first text engine (beside [QwtPlainTextEngine](#)) is returned, where [QwtTextEngine::mightRender](#) returns true. If there is none [QwtPlainTextEngine](#) is returned.

If no text engine is registered for the format [QwtPlainTextEngine](#) is returned.

Parameters

<i>text</i>	Text, needed in case of AutoText
<i>format</i>	Text format

12.83.4.27 `QSize QwtText::textSize (const QFont & defaultFont = QFont ()) const`

Find the height for a given width

Parameters

<i>defaultFont</i>	Font, used for the calculation if the text has no font
--------------------	--

Returns

Calculated height

Returns the size, that is needed to render text

Parameters

<i>defaultFont</i>	Font of the text
--------------------	------------------

Returns

Caluclated size

12.83.4.28 QColor QwtText::usedColor (const QColor & *defaultColor*) const

Return the color of the text, if it has one. Otherwise return defaultColor.

Parameters

<i>defaultColor</i>	Default color
---------------------	---------------

See also

[setColor\(\)](#), [color\(\)](#), PaintAttributes

12.83.4.29 QFont QwtText::usedFont (const QFont & *defaultFont*) const

Return the font of the text, if it has one. Otherwise return defaultFont.

Parameters

<i>defaultFont</i>	Default font
--------------------	--------------

See also

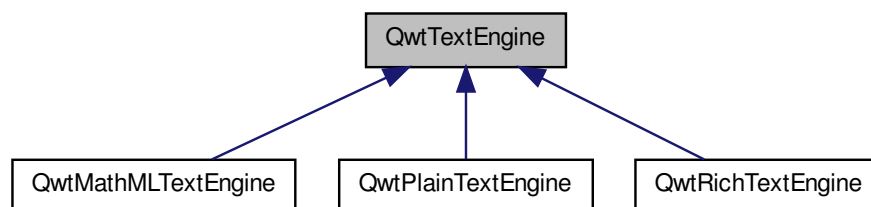
[setFont\(\)](#), [font\(\)](#), PaintAttributes

12.84 QwtTextEngine Class Reference

Abstract base class for rendering text strings.

```
#include <qwt_text_engine.h>
```

Inheritance diagram for QwtTextEngine:



Public Member Functions

- virtual void [draw](#) (QPainter *painter, const QRect &rect, int flags, const QString &text) const =0
- virtual int [heightForWidth](#) (const QFont &font, int flags, const QString &text, int width) const =0
- virtual bool [mightRender](#) (const QString &text) const =0
- virtual void [textMargins](#) (const QFont &font, const QString &text, int &left, int &right, int &top, int &bottom) const =0
- virtual QSize [textSize](#) (const QFont &font, int flags, const QString &text) const =0
- virtual [~QwtTextEngine](#) ()

Protected Member Functions

- [QwtTextEngine](#) ()

12.84.1 Detailed Description

Abstract base class for rendering text strings. A text engine is responsible for rendering texts for a specific text format. They are used by [QwtText](#) to render a text.

[QwtPlainTextEngine](#) and [QwtRichTextEngine](#) are part of the Qwt library.

[QwtMathMLTextEngine](#) can be found in Qwt MathML extension, that needs the MathML renderer of the Qt solutions package. Unfortunately it is only available with a commercial Qt license.

See also

[QwtText::setTextEngine\(\)](#)

12.84.2 Constructor & Destructor Documentation

12.84.2.1 QwtTextEngine::~QwtTextEngine () [virtual]

Destructor.

12.84.2.2 QwtTextEngine::QwtTextEngine () [protected]

Constructor.

12.84.3 Member Function Documentation

12.84.3.1 `virtual void QwtTextEngine::draw (QPainter * painter, const QRect & rect, int flags, const QString & text) const` **[pure virtual]**

Draw the text in a clipping rectangle

Parameters

<i>painter</i>	Painter
<i>rect</i>	Clipping rectangle
<i>flags</i>	Bitwise OR of the flags like in for QPainter::drawText
<i>text</i>	Text to be rendered

Implemented in [QwtPlainTextEngine](#), [QwtRichTextEngine](#), and [QwtMathMLTextEngine](#).

12.84.3.2 `virtual int QwtTextEngine::heightForWidth (const QFont & font, int flags, const QString & text, int width) const` **[pure virtual]**

Find the height for a given width

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags used like in QPainter::drawText
<i>text</i>	Text to be rendered
<i>width</i>	Width

Returns

Calculated height

Implemented in [QwtPlainTextEngine](#), [QwtRichTextEngine](#), and [QwtMathMLTextEngine](#).

12.84.3.3 `virtual bool QwtTextEngine::mightRender (const QString & text) const` **[pure virtual]**

Test if a string can be rendered by this text engine

Parameters

<i>text</i>	Text to be tested
-------------	-------------------

Returns

true, if it can be rendered

Implemented in [QwtPlainTextEngine](#), [QwtRichTextEngine](#), and [QwtMathMLTextEngine](#).

12.84.3.4 `virtual void QwtTextEngine::textMargins (const QFont & font,
const QString & text, int & left, int & right, int & top, int & bottom
) const [pure virtual]`

Return margins around the texts

The textSize might include margins around the text, like QFontMetrics::descent. In situations where texts need to be aligned in detail, knowing these margins might improve the layout calculations.

Parameters

<i>font</i>	Font of the text
<i>text</i>	Text to be rendered
<i>left</i>	Return value for the left margin
<i>right</i>	Return value for the right margin
<i>top</i>	Return value for the top margin
<i>bottom</i>	Return value for the bottom margin

Implemented in [QwtPlainTextEngine](#), [QwtRichTextEngine](#), and [QwtMathMLTextEngine](#).

12.84.3.5 `virtual QSize QwtTextEngine::textSize (const QFont & font, int
flags, const QString & text) const [pure virtual]`

Returns the size, that is needed to render text

Parameters

<i>font</i>	Font of the text
<i>flags</i>	Bitwise OR of the flags like in for QPainter::drawText
<i>text</i>	Text to be rendered

Returns

Calculated size

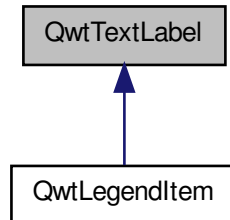
Implemented in [QwtPlainTextEngine](#), [QwtRichTextEngine](#), and [QwtMathMLTextEngine](#).

12.85 QwtTextLabel Class Reference

A Widget which displays a [QwtText](#).

```
#include <qwt_text_label.h>
```

Inheritance diagram for QwtTextLabel:



Public Slots

- void [clear](#) ()
- void [setText](#) (const QString &, [QwtText::TextFormat](#) textFormat=QwtText::AutoText)
- virtual void [setText](#) (const [QwtText](#) &)

Public Member Functions

- virtual int [heightForWidth](#) (int) const
- int [indent](#) () const
- int [margin](#) () const
- virtual QSize [minimumSizeHint](#) () const
- [QwtTextLabel](#) (const [QwtText](#) &, QWidget *parent=NULL)
- [QwtTextLabel](#) (QWidget *parent=NULL)
- void [setIndent](#) (int)
- void [setMargin](#) (int)
- virtual QSize [sizeHint](#) () const
- const [QwtText](#) & [text](#) () const
- QRect [textRect](#) () const
- virtual [~QwtTextLabel](#) ()

Protected Member Functions

- virtual void [drawContents](#) (QPainter *)
- virtual void [drawText](#) (QPainter *, const QRect &)
- virtual void [paintEvent](#) (QPaintEvent *e)

12.85.1 Detailed Description

A Widget which displays a [QwtText](#).

12.85.2 Constructor & Destructor Documentation

12.85.2.1 QwtTextLabel::QwtTextLabel (QWidget * *parent* = *NULL*) [explicit]

Constructs an empty label.

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.85.2.2 QwtTextLabel::QwtTextLabel (const QwtText & *text*, QWidget * *parent* = *NULL*) [explicit]

Constructs a label that displays the text, text

Parameters

<i>parent</i>	Parent widget
<i>text</i>	Text

12.85.2.3 QwtTextLabel::~~QwtTextLabel () [virtual]

Destructor.

12.85.3 Member Function Documentation

12.85.3.1 void QwtTextLabel::clear () [slot]

Clear the text and all [QwtText](#) attributes.

12.85.3.2 void QwtTextLabel::drawContents (QPainter * *painter*) [protected, virtual]

Redraw the text and focus indicator.

12.85.3.3 void QwtTextLabel::drawText (QPainter * *painter*, const QRect & *textRect*) [protected, virtual]

Redraw the text.

Reimplemented in [QwtLegendItem](#).

12.85.3.4 int QwtTextLabel::heightForWidth (int *width*) const [virtual]

Returns the preferred height for this widget, given the width.

Parameters

<i>width</i>	Width
--------------	-------

12.85.3.5 int QwtTextLabel::indent () const

Return label's text indent in pixels.

12.85.3.6 int QwtTextLabel::margin () const

Return label's text indent in pixels.

12.85.3.7 QSize QwtTextLabel::minimumSizeHint () const [virtual]

Return a minimum size hint.

12.85.3.8 void QwtTextLabel::paintEvent (QPaintEvent * *event*)
[protected, virtual]

Qt paint event

Parameters

<i>event</i>	Paint event
--------------	-------------

Reimplemented in [QwtLegendItem](#).

12.85.3.9 void QwtTextLabel::setIndent (int *indent*)

Set label's text indent in pixels

Parameters

<i>indent</i>	Indentation in pixels
---------------	-----------------------

12.85.3.10 void QwtTextLabel::setMargin (int *margin*)

Set label's margin in pixels

Parameters

<i>margin</i>	Margin in pixels
---------------	------------------

12.85.3.11 void QwtTextLabel::setText (const QwtText & *text*) [virtual, slot]

Change the label's text

Parameters

<i>text</i>	New text
-------------	----------

Reimplemented in [QwtLegendItem](#).**12.85.3.12 void QwtTextLabel::setText (const QString & *text*,
QwtText::TextFormat *textFormat* = *QwtText::AutoText*)
[slot]**Change the label's text, keeping all other [QwtText](#) attributes**Parameters**

<i>text</i>	New text
<i>textFormat</i>	Format of text

See also[QwtText](#)**12.85.3.13 QSize QwtTextLabel::sizeHint () const [virtual]**

Return label's margin in pixels.

Reimplemented in [QwtLegendItem](#).

12.85.3.14 const QwtText & QwtTextLabel::text () const

Return the text.

12.85.3.15 QRect QwtTextLabel::textRect () const

Calculate the rect for the text in widget coordinates

Returns

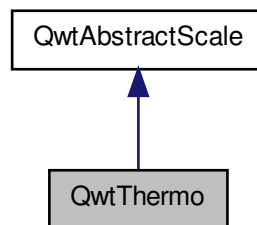
Text rect

12.86 QwtThermo Class Reference

The Thermometer Widget.

```
#include <qwt_thermo.h>
```

Inheritance diagram for QwtThermo:



Public Types

- enum **ScalePos** {
 NoScale,
 LeftScale,
 RightScale,
 TopScale,
 BottomScale }

Public Slots

- void [setValue](#) (double val)

Public Member Functions

- const QBrush & [alarmBrush](#) () const
- const QColor & [alarmColor](#) () const
- bool [alarmEnabled](#) () const
- double [alarmLevel](#) () const
- bool [autoScale](#) () const
- int [borderWidth](#) () const
- const QBrush & [fillBrush](#) () const
- const QColor & [fillColor](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const
- double [minValue](#) () const
- int [pipeWidth](#) () const
- [QwtThermo](#) (QWidget *parent=NULL)
- const [QwtScaleDraw](#) * [scaleDraw](#) () const
- const [QwtScaleEngine](#) * [scaleEngine](#) () const
- [QwtScaleEngine](#) * [scaleEngine](#) ()
- const [QwtScaleMap](#) & [scaleMap](#) () const
- int [scaleMaxMajor](#) () const
- int [scaleMaxMinor](#) () const
- ScalePos [scalePosition](#) () const
- void [setAlarmBrush](#) (const QBrush &b)
- void [setAlarmColor](#) (const QColor &c)
- void [setAlarmEnabled](#) (bool tf)
- void [setAlarmLevel](#) (double v)
- void [setAutoScale](#) ()
- void [setBorderWidth](#) (int w)
- void [setFillBrush](#) (const QBrush &b)
- void [setFillColor](#) (const QColor &c)
- void [setMargin](#) (int m)
- void [setMaxValue](#) (double v)
- void [setMinValue](#) (double v)
- void [setOrientation](#) (Qt::Orientation o, ScalePos s)
- void [setPipeWidth](#) (int w)
- void [setRange](#) (double vmin, double vmax, bool lg=false)
- void [setScale](#) (double vmin, double vmax, double step=0.0)
- void [setScale](#) (const [QwtScaleDiv](#) &s)
- void [setScale](#) (const [QwtDoubleInterval](#) &, double step=0.0)
- void [setScaleDraw](#) ([QwtScaleDraw](#) *)
- void [setScaleEngine](#) ([QwtScaleEngine](#) *)
- void [setScaleMaxMajor](#) (int ticks)

- void [setScaleMaxMinor](#) (int ticks)
- void [setScalePosition](#) (ScalePos s)
- virtual QSize [sizeHint](#) () const
- double [value](#) () const
- virtual [~QwtThermo](#) ()

Protected Member Functions

- const [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) () const
- [QwtAbstractScaleDraw](#) * [abstractScaleDraw](#) ()
- void [draw](#) (QPainter *p, const QRect &update_rect)
- void [drawThermo](#) (QPainter *p)
- virtual void [fontChange](#) (const QFont &oldFont)
- void [layoutThermo](#) (bool update=true)
- virtual void [paintEvent](#) (QPaintEvent *e)
- void [rescale](#) (double vmin, double vmax, double step=0.0)
- virtual void [resizeEvent](#) (QResizeEvent *e)
- virtual void [scaleChange](#) ()
- [QwtScaleDraw](#) * [scaleDraw](#) ()
- void [setAbstractScaleDraw](#) ([QwtAbstractScaleDraw](#) *)

12.86.1 Detailed Description

The Thermometer Widget. [QwtThermo](#) is a widget which displays a value in an interval. It supports:

- a horizontal or vertical layout;
- a range;
- a scale;
- an alarm level.

By default, the scale and range run over the same interval of values. [QwtAbstractScale::setScale\(\)](#) changes the interval of the scale and allows easy conversion between physical units.

The example shows how to make the scale indicate in degrees Fahrenheit and to set the value in degrees Kelvin:

```
#include <qapplication.h>
#include <qwt_thermo.h>

double Kelvin2Fahrenheit(double kelvin)
{
    // see http://en.wikipedia.org/wiki/Kelvin
    return 1.8*kelvin - 459.67;
}

int main(int argc, char **argv)
{
```

```

const double minKelvin = 0.0;
const double maxKelvin = 500.0;

QApplication a(argc, argv);
QwtThermo t;
t.setRange(minKelvin, maxKelvin);
t.setScale(Kelvin2Fahrenheit(minKelvin), Kelvin2Fahrenheit(maxKelvin));
// set the value in Kelvin but the scale displays in Fahrenheit
// 273.15 Kelvin = 0 Celsius = 32 Fahrenheit
t.setValue(273.15);
a.setMainWidget(&t);
t.show();
return a.exec();
}

```

12.86.2 Constructor & Destructor Documentation

12.86.2.1 QwtThermo::QwtThermo (QWidget * *parent* = *NULL*) [explicit]

Constructor

Parameters

<i>parent</i>	Parent widget
---------------	---------------

12.86.2.2 QwtThermo::~~QwtThermo () [virtual]

Destructor.

12.86.3 Member Function Documentation

12.86.3.1 const QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () const [protected, inherited]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.86.3.2 QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw () [protected, inherited]

Returns

Scale draw

See also

[setAbstractScaleDraw\(\)](#)

12.86.3.3 const QBrush & QwtThermo::alarmBrush () const

Return the liquid brush above the alarm threshold.

See also

[setAlarmBrush\(\)](#)

12.86.3.4 const QColor & QwtThermo::alarmColor () const

Return the liquid color above the alarm threshold.

12.86.3.5 bool QwtThermo::alarmEnabled () const

Return if the alarm threshold is enabled or disabled.

12.86.3.6 double QwtThermo::alarmLevel () const

Return the alarm threshold.

See also

[setAlarmLevel\(\)](#)

12.86.3.7 bool QwtAbstractScale::autoScale () const [inherited]**Returns**

true if autoscaling is enabled

12.86.3.8 int QwtThermo::borderWidth () const

Return the border width of the thermometer pipe.

See also

[setBorderWidth\(\)](#)

**12.86.3.9 void QwtThermo::draw (QPainter * *painter*, const QRect & *rect*)
[protected]**

Draw the whole [QwtThermo](#).

Parameters

<i>painter</i>	Painter
<i>rect</i>	Update rectangle

**12.86.3.10 void QwtThermo::drawThermo (QPainter * *painter*)
[protected]**

Redraw the liquid in thermometer pipe.

Parameters

<i>painter</i>	Painter
----------------	---------

12.86.3.11 const QBrush & QwtThermo::fillBrush () const

Return the liquid brush.

See also

[setFillBrush\(\)](#)

12.86.3.12 const QColor & QwtThermo::fillColor () const

Return the liquid color.

See also

[setFillColor\(\)](#)

**12.86.3.13 void QwtThermo::fontChange (const QFont & *oldFont*)
[protected, virtual]**

Notify a font change.

12.86.3.14 `void QwtThermo::layoutThermo (bool update_geometry = true)`
[protected]

Recalculate the [QwtThermo](#) geometry and layout based on the `QwtThermo::rect()` and the fonts.

Parameters

<i>update_geometry</i>	notify the layout system and call update to redraw the scale
------------------------	--

12.86.3.15 `double QwtThermo::maxValue () const`

Return the maximum value.

12.86.3.16 `QSize QwtThermo::minimumSizeHint () const` **[virtual]**

Return a minimum size hint.

Warning

The return value depends on the font and the scale.

See also

[sizeHint\(\)](#)

12.86.3.17 `double QwtThermo::minValue () const`

Return the minimum value.

12.86.3.18 `void QwtThermo::paintEvent (QPaintEvent * event)`
[protected, virtual]

Qt paint event. event Paint event

12.86.3.19 `int QwtThermo::pipeWidth () const`

Return the width of the pipe.

See also

[setPipeWidth\(\)](#)

12.86.3.20 `void QwtAbstractScale::rescale (double vmin, double vmax,
double stepSize = 0.0) [protected, inherited]`

Recalculate the scale division and update the scale draw.

Parameters

<i>vmin</i>	Lower limit of the scale interval
<i>vmax</i>	Upper limit of the scale interval
<i>stepSize</i>	Major step size

See also

[scaleChange\(\)](#)

12.86.3.21 `void QwtThermo::resizeEvent (QResizeEvent * e)
[protected, virtual]`

Qt resize event handler.

12.86.3.22 `void QwtThermo::scaleChange () [protected, virtual]`

Notify a scale change.

Reimplemented from [QwtAbstractScale](#).

12.86.3.23 `QwtScaleDraw * QwtThermo::scaleDraw () [protected]`

Returns

the scale draw of the thermo

See also

[setScaleDraw\(\)](#)

12.86.3.24 `const QwtScaleDraw * QwtThermo::scaleDraw () const`

Returns

the scale draw of the thermo

See also

[setScaleDraw\(\)](#)

12.86.3.25 `const QwtScaleEngine * QwtAbstractScale::scaleEngine () const`
[[inherited](#)]

Returns

Scale engine

See also

[setScaleEngine\(\)](#)

12.86.3.26 `QwtScaleEngine * QwtAbstractScale::scaleEngine ()`
[[inherited](#)]

Returns

Scale engine

See also

[setScaleEngine\(\)](#)

12.86.3.27 `const QwtScaleMap & QwtAbstractScale::scaleMap () const`
[[inherited](#)]

Returns

[abstractScaleDraw\(\)->scaleMap\(\)](#)

12.86.3.28 `int QwtAbstractScale::scaleMaxMajor () const` [[inherited](#)]

Returns

Max. number of major tick intervals The default value is 5.

12.86.3.29 `int QwtAbstractScale::scaleMaxMinor () const` [inherited]

Returns

Max. number of minor tick intervals The default value is 3.

12.86.3.30 `QwtThermo::ScalePos QwtThermo::scalePosition () const`

Return the scale position.

See also

[setScalePosition\(\)](#)

12.86.3.31 `void QwtAbstractScale::setAbstractScaleDraw (
QwtAbstractScaleDraw * scaleDraw)` [protected,
inherited]

Set a scale draw.

scaleDraw has to be created with `new` and will be deleted in `~QwtAbstractScale` or the next call of `setAbstractScaleDraw`.

12.86.3.32 `void QwtThermo::setAlarmBrush (const QBrush & brush)`

Specify the liquid brush above the alarm threshold.

Parameters

<i>brush</i>	New brush. The default is solid white.
--------------	--

See also

[alarmBrush\(\)](#)

12.86.3.33 `void QwtThermo::setAlarmColor (const QColor & c)`

Specify the liquid color above the alarm threshold.

Parameters

<i>c</i>	New color. The default is white.
----------	----------------------------------

12.86.3.34 void QwtThermo::setAlarmEnabled (bool *tf*)

Enable or disable the alarm threshold.

Parameters

<i>tf</i> true (disabled) or false (enabled)
--

12.86.3.35 void QwtThermo::setAlarmLevel (double *level*)

Specify the alarm threshold.

Parameters

<i>level</i> Alarm threshold

See also

[alarmLevel\(\)](#)

12.86.3.36 void QwtAbstractScale::setAutoScale () [inherited]

Advise the widget to control the scale range internally.

Autoscaling is on by default.

See also

[setScale\(\)](#), [autoScale\(\)](#)

12.86.3.37 void QwtThermo::setBorderWidth (int *width*)

Set the border width of the pipe.

Parameters

<i>width</i> Border width

See also

[borderWidth\(\)](#)

12.86.3.38 void QwtThermo::setFillBrush (const QBrush & *brush*)

Change the brush of the liquid.

Parameters

<i>brush</i>	New brush. The default brush is solid black.
--------------	--

See also

[fillBrush\(\)](#)

12.86.3.39 void QwtThermo::setFillColor (const QColor & *c*)

Change the color of the liquid.

Parameters

<i>c</i>	New color. The default color is black.
----------	--

See also

[fillColor\(\)](#)

12.86.3.40 void QwtThermo::setMargin (int *m*)

Specify the distance between the pipe's endpoints and the widget's border.

The margin is used to leave some space for the scale labels. If a large font is used, it is advisable to adjust the margins.

Parameters

<i>m</i>	New Margin. The default values are 10 for horizontal orientation and 20 for vertical orientation.
----------	---

Warning

The margin has no effect if the scale is disabled.

This function is a NOOP because margins are determined automatically.

12.86.3.41 void QwtThermo::setMaxValue (double *max*)

Set the maximum value.

Parameters

<i>max</i>	Maximum value
------------	---------------

See also

[maxValue\(\)](#), [setMinValue\(\)](#)

12.86.3.42 void QwtThermo::setMinValue (double *min*)

Set the minimum value.

Parameters

<i>min</i>	Minimum value
------------	---------------

See also

[minValue\(\)](#), [setMaxValue\(\)](#)

12.86.3.43 void QwtThermo::setOrientation (Qt::Orientation *o*, ScalePos *s*)

Set the thermometer orientation and the scale position.

The scale position NoScale disables the scale.

Parameters

<i>o</i>	orientation. Possible values are Qt::Horizontal and Qt::Vertical. The default value is Qt::Vertical.
<i>s</i>	Position of the scale. The default value is NoScale.

A valid combination of scale position and orientation is enforced:

- a horizontal thermometer can have the scale positions TopScale, BottomScale or NoScale;
- a vertical thermometer can have the scale positions LeftScale, RightScale or NoScale;
- an invalid scale position will default to NoScale.

See also

[setScalePosition\(\)](#)

12.86.3.44 void QwtThermo::setPipeWidth (int *width*)

Change the width of the pipe.

Parameters

<i>width</i>	Width of the pipe
--------------	-------------------

See also

[pipeWidth\(\)](#)

12.86.3.45 `void QwtThermo::setRange (double vmin, double vmax, bool logarithmic = false)`

Set the range.

Parameters

<i>vmin</i>	value corresponding lower or left end of the thermometer
<i>vmax</i>	value corresponding to the upper or right end of the thermometer
<i>logarithmic</i>	logarithmic mapping, true or false

12.86.3.46 `void QwtAbstractScale::setScale (const QwtDoubleInterval & interval, double stepSize = 0.0) [inherited]`

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>interval</i>	Interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.86.3.47 `void QwtAbstractScale::setScale (const QwtScaleDiv & scaleDiv) [inherited]`

Specify a scale.

Disable autoscaling and define a scale by a scale division

Parameters

<i>scaleDiv</i>	Scale division
-----------------	----------------

See also

[setAutoScale\(\)](#)

12.86.3.48 `void QwtAbstractScale::setScale (double vmin, double vmax,
double stepSize = 0.0) [inherited]`

Specify a scale.

Disable autoscaling and define a scale by an interval and a step size

Parameters

<i>vmin</i>	lower limit of the scale interval
<i>vmax</i>	upper limit of the scale interval
<i>stepSize</i>	major step size

See also

[setAutoScale\(\)](#)

12.86.3.49 `void QwtThermo::setScaleDraw (QwtScaleDraw * scaleDraw)`

Set a scale draw.

For changing the labels of the scales, it is necessary to derive from [QwtScaleDraw](#) and overload [QwtScaleDraw::label\(\)](#).

Parameters

<i>scaleDraw</i>	ScaleDraw object, that has to be created with new and will be deleted in ~QwtThermo or the next call of setScaleDraw() .
------------------	--

12.86.3.50 `void QwtAbstractScale::setScaleEngine (QwtScaleEngine *
scaleEngine) [inherited]`

Set a scale engine.

The scale engine is responsible for calculating the scale division, and in case of auto scaling how to align the scale.

scaleEngine has to be created with new and will be deleted in ~QwtAbstractScale or the next call of `setScaleEngine`.

12.86.3.51 void QwtAbstractScale::setScaleMaxMajor (int *ticks*)
[inherited]

Set the maximum number of major tick intervals.

The scale's major ticks are calculated automatically such that the number of major intervals does not exceed ticks. The default value is 5.

Parameters

<i>ticks</i> maximal number of major ticks.

See also

[QwtAbstractScaleDraw](#)

12.86.3.52 void QwtAbstractScale::setScaleMaxMinor (int *ticks*)
[inherited]

Set the maximum number of minor tick intervals.

The scale's minor ticks are calculated automatically such that the number of minor intervals does not exceed ticks. The default value is 3.

Parameters

<i>ticks</i>

See also

[QwtAbstractScaleDraw](#)

12.86.3.53 void QwtThermo::setScalePosition (ScalePos *scalePos*)

Change the scale position (and thermometer orientation).

Parameters

<i>scalePos</i> Position of the scale.
--

A valid combination of scale position and orientation is enforced:

- if the new scale position is LeftScale or RightScale, the scale orientation will become Qt::Vertical;
- if the new scale position is BottomScale or TopScale, the scale orientation will become Qt::Horizontal;

- if the new scale position is NoScale, the scale orientation will not change.

See also

[setOrientation\(\)](#), [scalePosition\(\)](#)

12.86.3.54 void QwtThermo::setValue (double *value*) [slot]

Set the current value.

Parameters

<i>value</i>	New Value
--------------	-----------

See also

[value\(\)](#)

12.86.3.55 QSize QwtThermo::sizeHint () const [virtual]

Returns

the minimum size hint

See also

[minimumSizeHint\(\)](#)

12.86.3.56 double QwtThermo::value () const

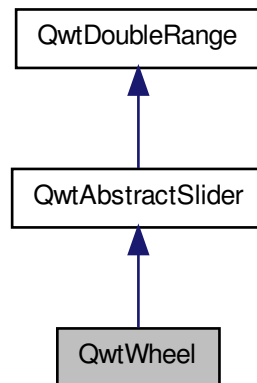
Return the value.

12.87 QwtWheel Class Reference

The Wheel Widget.

```
#include <qwt_wheel.h>
```

Inheritance diagram for QwtWheel:



Public Types

- enum [ScrollMode](#) {
 ScrNone,
 ScrMouse,
 ScrTimer,
 ScrDirect,
 ScrPage }

Public Slots

- virtual void [fitValue](#) (double val)
- virtual void [incValue](#) (int steps)
- virtual void [setReadOnly](#) (bool)
- virtual void [setValue](#) (double val)

Signals

- void [sliderMoved](#) (double value)
- void [sliderPressed](#) ()
- void [sliderReleased](#) ()
- void [valueChanged](#) (double value)

Public Member Functions

- virtual void [incPages](#) (int)
- int [internalBorder](#) () const
- bool [isReadOnly](#) () const
- bool [isValid](#) () const
- double [mass](#) () const
- double [maxValue](#) () const
- virtual QSize [minimumSizeHint](#) () const
- double [minValue](#) () const
- Qt::Orientation [orientation](#) () const
- int [pageSize](#) () const
- bool [periodic](#) () const
- [QwtWheel](#) (QWidget *parent=NULL)
- void [setInternalBorder](#) (int width)
- void [setMass](#) (double val)
- virtual void [setOrientation](#) (Qt::Orientation)
- void [setPeriodic](#) (bool tf)
- void [setRange](#) (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void [setStep](#) (double)
- void [setTickCnt](#) (int cnt)
- void [setTotalAngle](#) (double angle)
- void [setTracking](#) (bool enable)
- void [setUpdateTime](#) (int t)
- void [setValid](#) (bool valid)
- void [setViewAngle](#) (double angle)
- void [setWheelWidth](#) (int w)
- virtual QSize [sizeHint](#) () const
- double [step](#) () const
- void [stopMoving](#) ()
- int [tickCnt](#) () const
- double [totalAngle](#) () const
- double [value](#) () const
- double [viewAngle](#) () const
- virtual [~QwtWheel](#) ()

Protected Member Functions

- void [draw](#) (QPainter *, const QRect &)
- void [drawWheel](#) (QPainter *, const QRect &)
- void [drawWheelBackground](#) (QPainter *, const QRect &)
- double [exactPrevValue](#) () const
- double [exactValue](#) () const
- virtual void [getScrollMode](#) (const QPoint &, int &scrollMode, int &direction)
- virtual double [getValue](#) (const QPoint &)
- virtual void [keyPressEvent](#) (QKeyEvent *e)

- void [layoutWheel](#) (bool update=true)
- virtual void [mouseMoveEvent](#) (QMouseEvent *e)
- double [mouseOffset](#) () const
- virtual void [mousePressEvent](#) (QMouseEvent *e)
- virtual void [mouseReleaseEvent](#) (QMouseEvent *e)
- virtual void [paintEvent](#) (QPaintEvent *e)
- virtual void [paletteChange](#) (const QPalette &)
- double [prevValue](#) () const
- virtual void [rangeChange](#) ()
- virtual void [resizeEvent](#) (QResizeEvent *e)
- int [scrollMode](#) () const
- void [setColorArray](#) ()
- void [setMouseOffset](#) (double)
- virtual void [setPosition](#) (const QPoint &)
- virtual void [stepChange](#) ()
- virtual void [timerEvent](#) (QTimerEvent *e)
- virtual void [valueChange](#) ()
- virtual void [wheelEvent](#) (QWheelEvent *e)

12.87.1 Detailed Description

The Wheel Widget. The wheel widget can be used to change values over a very large range in very small steps. Using the `setMass` member, it can be configured as a fly-wheel.

See also

The radio example.

12.87.2 Member Enumeration Documentation

12.87.2.1 enum QwtAbstractSlider::ScrollMode **[inherited]**

Scroll mode

See also

[getScrollMode\(\)](#)

12.87.3 Constructor & Destructor Documentation

12.87.3.1 QwtWheel::QwtWheel (QWidget * *parent* = *NULL*) **[explicit]**

Constructor.

12.87.3.2 QwtWheel::~QwtWheel () [virtual]

Destructor.

12.87.4 Member Function Documentation**12.87.4.1 void QwtWheel::draw (QPainter * *painter*, const QRect &) [protected]**

Redraw panel and wheel

Parameters

<i>painter</i>	Painter
----------------	---------

12.87.4.2 void QwtWheel::drawWheel (QPainter * *painter*, const QRect & *r*) [protected]

Redraw the wheel.

Parameters

<i>painter</i>	painter
<i>r</i>	contents rectangle

12.87.4.3 void QwtWheel::drawWheelBackground (QPainter * *painter*, const QRect & *r*) [protected]

Draw the Wheel's background gradient

Parameters

<i>painter</i>	Painter
<i>r</i>	Bounding rectangle

12.87.4.4 double QwtDoubleRange::exactPrevValue () const [protected, inherited]

Returns the exact previous value.

12.87.4.5 double QwtDoubleRange::exactValue () const [protected, inherited]

Returns the exact value.

The exact value is the value which [QwtDoubleRange::value](#) would return if the value were not adjusted to the step raster. It differs from the current value only if [QwtDoubleRange::fitValue](#) or [QwtDoubleRange::incValue](#) have been used before. This function is intended for internal use in derived classes.

12.87.4.6 void QwtAbstractSlider::fitValue (double *value*) [virtual, slot, inherited]

Set the slider's value to the nearest integer multiple of the step size.

Parameters

<i>value</i>	Value
--------------	-------

See also

[setValue\(\)](#), [incValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.87.4.7 void QwtWheel::getScrollMode (const QPoint & *p*, int & *scrollMode*, int & *direction*) [protected, virtual]

Determine the scrolling mode and direction corresponding to a specified point.

Parameters

<i>p</i>	point
<i>scrollMode</i>	scrolling mode
<i>direction</i>	direction

Implements [QwtAbstractSlider](#).

12.87.4.8 double QwtWheel::getValue (const QPoint & *p*) [protected, virtual]

Determine the value corresponding to a specified point.

Implements [QwtAbstractSlider](#).

12.87.4.9 `void QwtDoubleRange::incPages (int nPages) [virtual, inherited]`

Increment the value by a specified number of pages.

Parameters

<i>nPages</i>	Number of pages to increment. A negative number decrements the value.
---------------	---

Warning

The Page size is specified in the constructor.

12.87.4.10 `void QwtAbstractSlider::incValue (int steps) [virtual, slot, inherited]`

Increment the value by a specified number of steps.

Parameters

<i>steps</i>	number of steps
--------------	-----------------

See also

[setValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.87.4.11 `int QwtWheel::internalBorder () const`

Returns

Internal border width of the wheel.

See also

[setInternalBorder\(\)](#)

12.87.4.12 `bool QwtAbstractSlider::isReadOnly () const [inherited]`

In read only mode the slider can't be controlled by mouse or keyboard.

Returns

true if read only

See also

[setReadOnly\(\)](#)

12.87.4.13 `bool QwtAbstractSlider::isValid () const` [`inline`,
`inherited`]

See also

`QwtDblRange::isValid()`

Reimplemented from [QwtDoubleRange](#).

12.87.4.14 `void QwtAbstractSlider::keyPressEvent (QKeyEvent * e)`
[`protected`, `virtual`, `inherited`]

Handles key events

- `Key_Down`, `Key_Left`
Decrement by 1
- `Key_Up`, `Key_Right`
Increment by 1

Parameters

<i>e</i> Key event

See also

[isReadOnly\(\)](#)

Reimplemented in [QwtCompass](#), and [QwtDial](#).

12.87.4.15 `void QwtWheel::layoutWheel (bool update = true)`
[`protected`]

Recalculate the slider's geometry and layout based on.

12.87.4.16 double QwtWheel::mass () const [virtual]**Returns**

mass

Reimplemented from [QwtAbstractSlider](#).

12.87.4.17 double QwtDoubleRange::maxValue () const [inherited]

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in [QwtDoubleRange::setRange](#).

See also

[setRange\(\)](#)

12.87.4.18 QSize QwtWheel::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

The return value is based on the wheel width.

12.87.4.19 double QwtDoubleRange::minValue () const [inherited]

Returns the value at the first border of the range.

minValue returns the value which has been specified as the first parameter in [setRange\(\)](#).

See also

[setRange\(\)](#)

**12.87.4.20 void QwtAbstractSlider::mouseMoveEvent (QMouseEvent * e)
[protected, virtual, inherited]**

Mouse Move Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.87.4.21 `void QwtAbstractSlider::mousePressEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse press event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.87.4.22 `void QwtAbstractSlider::mouseReleaseEvent (QMouseEvent * e)`
[protected, virtual, inherited]

Mouse Release Event handler

Parameters

<i>e</i>	Mouse event
----------	-------------

12.87.4.23 `Qt::Orientation QwtAbstractSlider::orientation () const`
[inherited]

Returns

Orientation

See also

[setOrientation\(\)](#)

12.87.4.24 `int QwtDoubleRange::pageSize () const` **[inherited]**

Returns the page size in steps.

12.87.4.25 `void QwtWheel::paintEvent (QPaintEvent * e)` **[protected, virtual]**

Qt Paint Event.

12.87.4.26 `void QwtWheel::paletteChange (const QPalette &)`
[protected, virtual]

Call update() when the palette changes.

12.87.4.27 `bool QwtDoubleRange::periodic () const` [inherited]

Returns true if the range is periodic.

See also

[setPeriodic\(\)](#)

12.87.4.28 `double QwtDoubleRange::prevValue () const` [protected, inherited]

Returns the previous value.

12.87.4.29 `void QwtDoubleRange::rangeChange ()` [protected, virtual, inherited]

Notify a change of the range.

This virtual function is called whenever the range changes. The default implementation does nothing.

Reimplemented in [QwtCounter](#), [QwtDial](#), and [QwtSlider](#).

12.87.4.30 `void QwtWheel::resizeEvent (QResizeEvent * e)` [protected, virtual]

Qt Resize Event.

12.87.4.31 `void QwtWheel::setColorArray ()` [protected]

Set up the color array for the background pixmap.

12.87.4.32 void QwtWheel::setInternalBorder (int *w*)

Set the internal border width of the wheel.

The internal border must not be smaller than 1 and is limited in dependence on the wheel's size. Values outside the allowed range will be clipped.

The internal border defaults to 2.

Parameters

<i>w</i>	border width
----------	--------------

See also

[internalBorder\(\)](#)

12.87.4.33 void QwtWheel::setMass (double *val*) [virtual]

Set the mass of the wheel.

Assigning a mass turns the wheel into a flywheel.

Parameters

<i>val</i>	the wheel's mass
------------	------------------

Reimplemented from [QwtAbstractSlider](#).

12.87.4.34 void QwtWheel::setOrientation (Qt::Orientation *o*) [virtual]

Set the wheel's orientation.

Parameters

<i>o</i>	Orientation. Allowed values are Qt::Horizontal and Qt::Vertical. Defaults to Qt::Horizontal.
----------	--

See also

[QwtAbstractSlider::orientation\(\)](#)

Reimplemented from [QwtAbstractSlider](#).

12.87.4.35 void QwtDoubleRange::setPeriodic (bool *tf*) [inherited]

Make the range periodic.

When the range is periodic, the value will be set to a point inside the interval such that

```
point = value + n * width
```

if the user tries to set a new value which is outside the range. If the range is nonperiodic (the default), values outside the range will be clipped.

Parameters

<i>tf</i>	true for a periodic range
-----------	---------------------------

12.87.4.36 void QwtAbstractSlider::setPosition (const QPoint & p)
[protected, virtual, inherited]

Move the slider to a specified point, adjust the value and emit signals if necessary.

12.87.4.37 void QwtDoubleRange::setRange (double vmin, double vmax,
double vstep = 0.0, int pageSize = 1) [inherited]

Specify range and step size.

Parameters

<i>vmin</i>	lower boundary of the interval
<i>vmax</i>	higher boundary of the interval
<i>vstep</i>	step width
<i>pageSize</i>	page size in steps

Warning

- A change of the range changes the value if it lies outside the new range. The current value will *not* be adjusted to the new step raster.
- $vmax < vmin$ is allowed.
- If the step size is left out or set to zero, it will be set to 1/100 of the interval length.
- If the step size has an absurd value, it will be corrected to a better one.

12.87.4.38 void QwtAbstractSlider::setReadOnly (bool readOnly)
[virtual, slot, inherited]

En/Disable read only mode

In read only mode the slider can't be controlled by mouse or keyboard.

Parameters

<i>readOnly</i>	Enables in case of true
-----------------	-------------------------

See also[isReadOnly\(\)](#)**12.87.4.39 void QwtDoubleRange::setStep (double *vstep*) [inherited]**

Change the step raster.

Parameters

<i>vstep</i>	new step width
--------------	----------------

Warning

The value will *not* be adjusted to the new step raster.

Reimplemented in [QwtCounter](#).

12.87.4.40 void QwtWheel::setTickCnt (int *cnt*)

Adjust the number of grooves in the wheel's surface.

The number of grooves is limited to $6 \leq cnt \leq 50$. Values outside this range will be clipped. The default value is 10.

Parameters

<i>cnt</i>	Number of grooves per 360 degrees
------------	-----------------------------------

See also[tickCnt\(\)](#)**12.87.4.41 void QwtWheel::setTotalAngle (double *angle*)**

Set the total angle which the wheel can be turned.

One full turn of the wheel corresponds to an angle of 360 degrees. A total angle of $n \cdot 360$ degrees means that the wheel has to be turned n times around its axis to get from the minimum value to the maximum value.

The default setting of the total angle is 360 degrees.

Parameters

<i>angle</i>	total angle in degrees
--------------	------------------------

See also[totalAngle\(\)](#)**12.87.4.42 void QwtAbstractSlider::setTracking (bool *enable*)
[inherited]**

Enables or disables tracking.

If tracking is enabled, the slider emits a [valueChanged\(\)](#) signal whenever its value changes (the default behaviour). If tracking is disabled, the value changed() signal will only be emitted if:

- the user releases the mouse button and the value has changed or
- at the end of automatic scrolling.

Tracking is enabled by default.

Parameters

<i>enable</i>	true (enable) or false (disable) tracking.
---------------	--

12.87.4.43 void QwtAbstractSlider::setUpdateTime (int *t*) [inherited]

Specify the update interval for automatic scrolling.

Parameters

<i>t</i>	update interval in milliseconds
----------	---------------------------------

See also[getScrollMode\(\)](#)**12.87.4.44 void QwtAbstractSlider::setValid (bool *valid*) [inline, inherited]****Parameters**

<i>valid</i>	true/false
--------------	------------

See also

[QwtDbtRange::isValid\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.87.4.45 void QwtAbstractSlider::setValue (double *val*) [virtual, slot, inherited]

Move the slider to a specified value.

This function can be used to move the slider to a value which is not an integer multiple of the step size.

Parameters

<i>val</i> new value

See also

[fitValue\(\)](#)

Reimplemented from [QwtDoubleRange](#).

12.87.4.46 void QwtWheel::setViewAngle (double *angle*)

Specify the visible portion of the wheel.

You may use this function for fine-tuning the appearance of the wheel. The default value is 175 degrees. The value is limited from 10 to 175 degrees.

Parameters

<i>angle</i> Visible angle in degrees

See also

[viewAngle\(\)](#), [setTotalAngle\(\)](#)

12.87.4.47 void QwtWheel::setWheelWidth (int *w*)

Set the width of the wheel.

Corresponds to the wheel height for horizontal orientation, and the wheel width for vertical orientation.

Parameters

<i>w</i>	the wheel's width
----------	-------------------

12.87.4.48 QSize QwtWheel::sizeHint () const [virtual]**Returns**

a size hint

12.87.4.49 void QwtAbstractSlider::sliderMoved (double *value*) [signal, inherited]

This signal is emitted when the user moves the slider with the mouse.

Parameters

<i>value</i>	new value
--------------	-----------

12.87.4.50 void QwtAbstractSlider::sliderPressed () [signal, inherited]

This signal is emitted when the user presses the movable part of the slider (start ScrMouse Mode).

12.87.4.51 void QwtAbstractSlider::sliderReleased () [signal, inherited]

This signal is emitted when the user releases the movable part of the slider.

12.87.4.52 double QwtDoubleRange::step () const [inherited]**Returns**

the step size

See also

[setStep\(\)](#), [setRange\(\)](#)

Reimplemented in [QwtCounter](#).

12.87.4.53 `void QwtDoubleRange::stepChange () [protected, virtual, inherited]`

Notify a change of the step size.

This virtual function is called whenever the step size changes. The default implementation does nothing.

12.87.4.54 `void QwtAbstractSlider::stopMoving () [inherited]`

Stop updating if automatic scrolling is active.

12.87.4.55 `int QwtWheel::tickCnt () const`

Returns

Number of grooves in the wheel's surface.

See also

[setTickCnt\(\)](#)

12.87.4.56 `void QwtAbstractSlider::timerEvent (QTimerEvent * e) [protected, virtual, inherited]`

Qt timer event

Parameters

<i>e</i>	Timer event
----------	-------------

12.87.4.57 `double QwtWheel::totalAngle () const`

Returns

Total angle which the wheel can be turned.

See also

[setTotalAngle\(\)](#)

12.87.4.58 double QwtDoubleRange::value () const [inherited]

Returns the current value.

Reimplemented in [QwtCounter](#).

12.87.4.59 void QwtWheel::valueChange () [protected, virtual]

Notify value change.

Reimplemented from [QwtAbstractSlider](#).

12.87.4.60 void QwtAbstractSlider::valueChanged (double *value*) [signal, inherited]

Notify a change of value.

In the default setting (tracking enabled), this signal will be emitted every time the value changes (see [setTracking\(\)](#)).

Parameters

<i>value</i>	new value
--------------	-----------

12.87.4.61 double QwtWheel::viewAngle () const**Returns**

Visible portion of the wheel

See also

[setViewAngle\(\)](#), [totalAngle\(\)](#)

12.87.4.62 void QwtAbstractSlider::wheelEvent (QWheelEvent * *e*) [protected, virtual, inherited]

Wheel Event handler

Parameters

<i>e</i>	Wheel event
----------	-------------

Index

- ~QwtAbstractScale
 - QwtAbstractScale, [24](#)
- ~QwtAbstractScaleDraw
 - QwtAbstractScaleDraw, [32](#)
- ~QwtAbstractSlider
 - QwtAbstractSlider, [40](#)
- ~QwtAlphaColorMap
 - QwtAlphaColorMap, [53](#)
- ~QwtAnalogClock
 - QwtAnalogClock, [61](#)
- ~QwtArrowButton
 - QwtArrowButton, [89](#)
- ~QwtColorMap
 - QwtColorMap, [94](#)
- ~QwtCompass
 - QwtCompass, [101](#)
- ~QwtCounter
 - QwtCounter, [137](#)
- ~QwtCurveFitter
 - QwtCurveFitter, [150](#)
- ~QwtData
 - QwtData, [152](#)
- ~QwtDial
 - QwtDial, [159](#)
- ~QwtDialNeedle
 - QwtDialNeedle, [182](#)
- ~QwtDoubleRange
 - QwtDoubleRange, [206](#)
- ~QwtDynGridLayout
 - QwtDynGridLayout, [213](#)
- ~QwtEventPattern
 - QwtEventPattern, [223](#)
- ~QwtIntervalData
 - QwtIntervalData, [228](#)
- ~QwtKnob
 - QwtKnob, [233](#)
- ~QwtLegend
 - QwtLegend, [254](#)
- ~QwtLegendItem
 - QwtLegendItem, [261](#)
- ~QwtLegendItemManager
 - QwtLegendItemManager, [270](#)
- ~QwtLinearColorMap
 - QwtLinearColorMap, [274](#)
- ~QwtMagnifier
 - QwtMagnifier, [293](#)
- ~QwtMathMLTextEngine
 - QwtMathMLTextEngine, [301](#)
- ~QwtPanner
 - QwtPanner, [312](#)
- ~QwtPicker
 - QwtPicker, [326](#)
- ~QwtPickerMachine
 - QwtPickerMachine, [353](#)
- ~QwtPlainTextEngine
 - QwtPlainTextEngine, [356](#)
- ~QwtPlot
 - QwtPlot, [364](#)
- ~QwtPlotCanvas
 - QwtPlotCanvas, [387](#)
- ~QwtPlotCurve
 - QwtPlotCurve, [397](#)
- ~QwtPlotDict
 - QwtPlotDict, [420](#)
- ~QwtPlotGrid
 - QwtPlotGrid, [424](#)
- ~QwtPlotItem
 - QwtPlotItem, [441](#)
- ~QwtPlotLayout
 - QwtPlotLayout, [452](#)
- ~QwtPlotMagnifier
 - QwtPlotMagnifier, [461](#)
- ~QwtPlotMarker
 - QwtPlotMarker, [473](#)
- ~QwtPlotPanner
 - QwtPlotPanner, [489](#)
- ~QwtPlotPicker
 - QwtPlotPicker, [505](#)
- ~QwtPlotPrintFilter
 - QwtPlotPrintFilter, [529](#)
- ~QwtPlotRasterItem
 - QwtPlotRasterItem, [535](#)
- ~QwtPlotRescaler
 - QwtPlotRescaler, [549](#)
- ~QwtPlotScaleItem
 - QwtPlotScaleItem, [558](#)
- ~QwtPlotSpectrogram
 - QwtPlotSpectrogram, [576](#)
- ~QwtPlotSvgItem
 - QwtPlotSvgItem, [596](#)
- ~QwtRasterData
 - QwtRasterData, [648](#)
- ~QwtRoundScaleDraw
 - QwtRoundScaleDraw, [656](#)

- ~QwtScaleDraw
 - QwtScaleDraw, [673](#)
- ~QwtScaleEngine
 - QwtScaleEngine, [687](#)
- ~QwtScaleMap
 - QwtScaleMap, [693](#)
- ~QwtScaleTransformation
 - QwtScaleTransformation, [697](#)
- ~QwtScaleWidget
 - QwtScaleWidget, [700](#)
- ~QwtSpline
 - QwtSpline, [740](#)
- ~QwtSplineCurveFitter
 - QwtSplineCurveFitter, [744](#)
- ~QwtSymbol
 - QwtSymbol, [747](#)
- ~QwtText
 - QwtText, [754](#)
- ~QwtTextEngine
 - QwtTextEngine, [763](#)
- ~QwtTextLabel
 - QwtTextLabel, [767](#)
- ~QwtThermo
 - QwtThermo, [773](#)
- ~QwtWheel
 - QwtWheel, [789](#)
- abstractScaleDraw
 - QwtAbstractScale, [25](#)
 - QwtKnob, [233](#)
 - QwtSlider, [717](#), [718](#)
 - QwtThermo, [773](#)
- accept
 - QwtPicker, [327](#)
 - QwtPlotPicker, [506](#)
 - QwtPlotZoomer, [618](#)
- activate
 - QwtPlotLayout, [452](#)
- addColorStop
 - QwtLinearColorMap, [274](#)
- addItem
 - QwtDynGridLayout, [213](#)
- alarmBrush
 - QwtThermo, [774](#)
- alarmColor
 - QwtThermo, [774](#)
- alarmEnabled
 - QwtThermo, [774](#)
- alarmLevel
 - QwtThermo, [774](#)
- align
 - QwtLinearScaleEngine, [279](#)
- alignCanvasToScales
 - QwtPlotLayout, [452](#)
- alignLegend
 - QwtPlotLayout, [453](#)
- Alignment
 - QwtScaleDraw, [672](#)
- alignment
 - QwtScaleDraw, [673](#)
 - QwtScaleWidget, [701](#)
- alignScales
 - QwtPlotLayout, [453](#)
- alpha
 - QwtPlotRasterItem, [535](#)
 - QwtPlotSpectrogram, [577](#)
- append
 - QwtPicker, [327](#)
 - QwtPlotPicker, [506](#)
 - QwtPlotZoomer, [618](#)
- appended
 - QwtPicker, [327](#)
 - QwtPlotPicker, [507](#)
 - QwtPlotZoomer, [618](#), [619](#)
- apply
 - QwtPlotPrintFilter, [529](#)
- applyProperties
 - QwtPlot, [364](#)
- arrowSize
 - QwtArrowButton, [89](#)
- arrowType
 - QwtArrowButton, [89](#)
- aspectRatio
 - QwtPlotRescaler, [549](#)
- attach
 - QwtPlotCurve, [398](#)
 - QwtPlotGrid, [424](#)
 - QwtPlotItem, [441](#)
 - QwtPlotMarker, [473](#)
 - QwtPlotRasterItem, [535](#)
 - QwtPlotScaleItem, [559](#)
 - QwtPlotSpectrogram, [577](#)
 - QwtPlotSvgItem, [597](#)
- Attribute
 - QwtLinearScaleEngine, [279](#)
 - QwtLog10ScaleEngine, [286](#)
 - QwtScaleEngine, [687](#)
- attributes
 - QwtLinearScaleEngine, [280](#)
 - QwtLog10ScaleEngine, [287](#)

- QwtScaleEngine, 688
- autoDelete
 - QwtPlot, 364
 - QwtPlotDict, 420
- autoRefresh
 - QwtPlot, 364
- autoReplot
 - QwtPlot, 364
- autoScale
 - QwtAbstractScale, 25
 - QwtKnob, 234
 - QwtLinearScaleEngine, 280
 - QwtLog10ScaleEngine, 287
 - QwtScaleEngine, 688
 - QwtSlider, 718
 - QwtThermo, 774
- Axis
 - QwtPlot, 362
- axisAutoScale
 - QwtPlot, 364
- axisEnabled
 - QwtPlot, 365
- axisFont
 - QwtPlot, 365
- axisMaxMajor
 - QwtPlot, 365
- axisMaxMinor
 - QwtPlot, 365
- axisScaleDiv
 - QwtPlot, 366
- axisScaleDraw
 - QwtPlot, 366, 367
- axisScaleEngine
 - QwtPlot, 367, 368
- axisStepSize
 - QwtPlot, 368
- axisTitle
 - QwtPlot, 368
- axisValid
 - QwtPlot, 368
- axisWidget
 - QwtPlot, 369
- backgroundBrush
 - QwtText, 755
- backgroundPen
 - QwtText, 755
- baseline
 - QwtPlotCurve, 398
- begin
 - QwtPicker, 327
 - QwtPlotPicker, 507
 - QwtPlotZoomer, 619
- BGSTYLE
 - QwtSlider, 716
- bgStyle
 - QwtSlider, 718
- borderDistance
 - QwtPlotScaleItem, 559
- borderFlags
 - QwtDoubleInterval, 199
- BorderMode
 - QwtDoubleInterval, 198
- borderWidth
 - QwtKnob, 234
 - QwtSlider, 718
 - QwtThermo, 774
- boundingLabelRect
 - QwtScaleDraw, 673
- boundingRect
 - QwtAnalogClock, 61
 - QwtArrayData, 86
 - QwtCompass, 102
 - QwtCPointerData, 148
 - QwtData, 152
 - QwtDial, 159
 - QwtIntervalData, 228
 - QwtPlotCurve, 398
 - QwtPlotGrid, 425
 - QwtPlotItem, 441
 - QwtPlotMarker, 474
 - QwtPlotRasterItem, 536
 - QwtPlotScaleItem, 559
 - QwtPlotSpectrogram, 577
 - QwtPlotSvgItem, 597
 - QwtPolygonFData, 645
 - QwtRasterData, 648
- brush
 - QwtPlotCurve, 398
 - QwtSymbol, 747
- buildInterval
 - QwtLinearScaleEngine, 280
 - QwtLog10ScaleEngine, 287
 - QwtScaleEngine, 688
- buildNaturalSpline
 - QwtSpline, 740
- buildPeriodicSpline
 - QwtSpline, 740
- Button
 - QwtCounter, 137

- buttonReleased
 - QwtCounter, [138](#)
- CachePolicy
 - QwtPlotRasterItem, [533](#)
 - QwtPlotSpectrogram, [575](#)
- cachePolicy
 - QwtPlotRasterItem, [536](#)
 - QwtPlotSpectrogram, [577](#)
- canvas
 - QwtPlot, [369](#)
 - QwtPlotMagnifier, [461](#)
 - QwtPlotPanner, [490](#)
 - QwtPlotPicker, [508](#)
 - QwtPlotRescaler, [549](#)
 - QwtPlotZoomer, [619](#)
- canvasBackground
 - QwtPlot, [369](#)
- canvasLineWidth
 - QwtPlot, [370](#)
- canvasMap
 - QwtPlot, [370](#)
- canvasMargin
 - QwtPlotLayout, [453](#)
- canvasRect
 - QwtPlotLayout, [453](#)
- ceil125
 - QwtScaleArithmetic, [664](#)
- ceilEps
 - QwtScaleArithmetic, [664](#)
- center
 - QwtDialScaleDraw, [185](#)
 - QwtRoundScaleDraw, [656](#)
- changed
 - QwtPicker, [328](#)
 - QwtPlotPicker, [508](#)
 - QwtPlotZoomer, [619](#)
- checked
 - QwtLegendItem, [262](#)
- clear
 - QwtLegend, [254](#)
 - QwtLegendItem, [262](#)
 - QwtPlot, [370](#)
 - QwtTextLabel, [767](#)
- clicked
 - QwtLegendItem, [262](#)
- clipCircle
 - QwtClipper, [91](#)
- clipPolygon
 - QwtClipper, [92](#)
- clipPolygonF
 - QwtClipper, [92](#)
- clone
 - QwtSymbol, [747](#)
- closePolyline
 - QwtPlotCurve, [399](#)
- closestPoint
 - QwtPlotCurve, [399](#)
- coefficientsA
 - QwtSpline, [740](#)
- coefficientsB
 - QwtSpline, [740](#)
- coefficientsC
 - QwtSpline, [741](#)
- color
 - QwtAlphaColorMap, [53](#), [54](#)
 - QwtColorMap, [94](#)
 - QwtLinearColorMap, [274](#)
 - QwtPlotPrintFilter, [529](#)
 - QwtText, [755](#)
- color1
 - QwtLinearColorMap, [275](#)
- color2
 - QwtLinearColorMap, [275](#)
- colorIndex
 - QwtColorMap, [95](#)
 - QwtLinearColorMap, [275](#)
- colorMap
 - QwtPlotSpectrogram, [578](#)
- colorStops
 - QwtLinearColorMap, [275](#)
- colorTable
 - QwtAlphaColorMap, [54](#)
 - QwtColorMap, [95](#)
 - QwtLinearColorMap, [275](#)
- columnsForWidth
 - QwtDynGridLayout, [213](#)
- Command
 - QwtPickerClickPointMachine, [345](#)
 - QwtPickerClickRectMachine, [347](#)
 - QwtPickerDragPointMachine, [348](#)
 - QwtPickerDragRectMachine, [351](#)
 - QwtPickerMachine, [353](#)
 - QwtPickerPolygonMachine, [355](#)
- compareEps
 - QwtScaleArithmetic, [664](#)
- ConrecAttribute
 - QwtRasterData, [647](#)
- contains
 - QwtDoubleInterval, [199](#)

- QwtLinearScaleEngine, [280](#)
- QwtLog10ScaleEngine, [287](#)
- QwtScaleDiv, [667](#)
- QwtScaleEngine, [688](#)
- contentsRect
 - QwtAnalogClock, [61](#)
 - QwtCompass, [102](#)
 - QwtDial, [159](#)
- contentsWidget
 - QwtLegend, [254](#)
- contourLevels
 - QwtPlotSpectrogram, [578](#)
- contourLines
 - QwtRasterData, [648](#)
- contourPen
 - QwtPlotSpectrogram, [578](#)
- contourRasterSize
 - QwtPlotSpectrogram, [579](#)
- copy
 - QwtAlphaColorMap, [54](#)
 - QwtArrayData, [86](#)
 - QwtColorMap, [95](#)
 - QwtCPointerData, [148](#)
 - QwtData, [152](#)
 - QwtLinearColorMap, [276](#)
 - QwtPolygonFData, [645](#)
 - QwtRasterData, [648](#)
 - QwtScaleTransformation, [697](#)
- count
 - QwtDynGridLayout, [214](#)
- cursor
 - QwtPanner, [312](#)
 - QwtPlotPanner, [490](#)
- CurveAttribute
 - QwtPlotCurve, [395](#)
- curveFitter
 - QwtPlotCurve, [399](#)
- curvePen
 - QwtLegendItem, [262](#)
- CurveStyle
 - QwtPlotCurve, [395](#)
- CurveType
 - QwtPlotCurve, [395](#)
- curveType
 - QwtPlotCurve, [399](#)
- data
 - QwtPlotCurve, [400](#)
 - QwtPlotSpectrogram, [579](#)
 - QwtPolygonFData, [645](#)
- dataSize
 - QwtPlotCurve, [400](#)
- defaultContourPen
 - QwtPlotSpectrogram, [579](#)
- detach
 - QwtPlotCurve, [400](#)
 - QwtPlotGrid, [425](#)
 - QwtPlotItem, [442](#)
 - QwtPlotMarker, [474](#)
 - QwtPlotRasterItem, [536](#)
 - QwtPlotScaleItem, [559](#)
 - QwtPlotSpectrogram, [580](#)
 - QwtPlotSvgItem, [597](#)
- detachItems
 - QwtPlot, [370](#)
 - QwtPlotDict, [420](#)
- deviceClipping
 - QwtPainter, [306](#)
- deviceClipRect
 - QwtPainter, [306](#)
- dimForLength
 - QwtScaleWidget, [701](#)
- Direction
 - QwtAnalogClock, [60](#)
 - QwtCompass, [100](#)
 - QwtDial, [158](#)
- direction
 - QwtAnalogClock, [62](#)
 - QwtCompass, [102](#)
 - QwtDial, [159](#)
- discardRaster
 - QwtRasterData, [649](#)
- DisplayMode
 - QwtPicker, [321](#)
 - QwtPlotPicker, [500](#)
 - QwtPlotSpectrogram, [575](#)
 - QwtPlotZoomer, [611](#)
- displayPolicy
 - QwtLegend, [255](#)
- divideEps
 - QwtScaleArithmetic, [665](#)
- divideInterval
 - QwtLinearScaleEngine, [281](#)
 - QwtLog10ScaleEngine, [288](#)
 - QwtScaleEngine, [689](#)
- divideScale
 - QwtLinearScaleEngine, [281](#)
 - QwtLog10ScaleEngine, [288](#)
 - QwtScaleEngine, [689](#)
- draw

- QwtAbstractScaleDraw, 32
- QwtCompassMagnetNeedle, 128
- QwtCompassRose, 130
- QwtCompassWindArrow, 133
- QwtDialNeedle, 182
- QwtDialScaleDraw, 185
- QwtDialSimpleNeedle, 195
- QwtKnob, 234
- QwtMathMLTextEngine, 302
- QwtPlainTextEngine, 357
- QwtPlotCurve, 400, 401
- QwtPlotGrid, 425
- QwtPlotItem, 442
- QwtPlotMarker, 474
- QwtPlotRasterItem, 536
- QwtPlotScaleItem, 560
- QwtPlotSpectrogram, 580
- QwtPlotSvgItem, 597
- QwtRichTextEngine, 652
- QwtRoundScaleDraw, 656
- QwtScaleDraw, 674
- QwtScaleWidget, 701
- QwtSimpleCompassRose, 710
- QwtSlider, 718
- QwtSymbol, 747, 748
- QwtText, 755
- QwtTextEngine, 764
- QwtThermo, 775
- QwtWheel, 790
- drawArrow
 - QwtArrowButton, 89
- drawArrowNeedle
 - QwtDialSimpleNeedle, 195
- drawAt
 - QwtPlotMarker, 474
- drawBackbone
 - QwtAbstractScaleDraw, 32
 - QwtDialScaleDraw, 185
 - QwtRoundScaleDraw, 656
 - QwtScaleDraw, 674
- drawButtonLabel
 - QwtArrowButton, 89
- drawCanvas
 - QwtPlot, 371
 - QwtPlotCanvas, 387
- drawContents
 - QwtAnalogClock, 62
 - QwtCompass, 102
 - QwtDial, 159
 - QwtLegendItem, 262
 - QwtPlotCanvas, 387
 - QwtTextLabel, 767
- drawContourLines
 - QwtPlotSpectrogram, 580
- drawCurve
 - QwtPlotCurve, 402
- drawDots
 - QwtPlotCurve, 402
- drawEllipse
 - QwtPainter, 306
- drawFocusIndicator
 - QwtAnalogClock, 62
 - QwtCompass, 103
 - QwtDial, 160
 - QwtPlotCanvas, 388
- drawFrame
 - QwtAnalogClock, 63
 - QwtCompass, 103
 - QwtDial, 160
- drawHand
 - QwtAnalogClock, 63
- drawIdentifier
 - QwtLegendItem, 262
- drawItem
 - QwtLegendItem, 263
- drawItems
 - QwtPlot, 371
- drawKnob
 - QwtCompassMagnetNeedle, 128
 - QwtCompassWindArrow, 133
 - QwtDialNeedle, 182
 - QwtDialSimpleNeedle, 196
 - QwtKnob, 234
- drawLabel
 - QwtAbstractScaleDraw, 32
 - QwtDialScaleDraw, 186
 - QwtRoundScaleDraw, 657
 - QwtScaleDraw, 674
- drawLine
 - QwtPainter, 307
- drawLines
 - QwtPlotCurve, 403
- drawMarker
 - QwtKnob, 234
- drawNeedle
 - QwtAnalogClock, 63
 - QwtCompass, 103
 - QwtDial, 160
- drawPie
 - QwtPainter, 307

- drawPoint
 - QwtPainter, 307
- drawPointer
 - QwtCompassMagnetNeedle, 128
- drawPolygon
 - QwtPainter, 307
- drawPolyline
 - QwtPainter, 307
- drawRayNeedle
 - QwtDialSimpleNeedle, 196
- drawRect
 - QwtPainter, 307
- drawRose
 - QwtCompass, 103
 - QwtSimpleCompassRose, 710
- drawRoundFrame
 - QwtPainter, 308
- drawRubberBand
 - QwtPicker, 328
 - QwtPlotPicker, 508
 - QwtPlotZoomer, 620
- drawScale
 - QwtAnalogClock, 64
 - QwtCompass, 104
 - QwtDial, 161
- drawScaleContents
 - QwtAnalogClock, 64
 - QwtCompass, 104
 - QwtDial, 161
- drawSimpleRichText
 - QwtPainter, 308
- drawSlider
 - QwtSlider, 718
- drawSteps
 - QwtPlotCurve, 403
- drawSticks
 - QwtPlotCurve, 403
- drawStyle1Needle
 - QwtCompassWindArrow, 133
- drawStyle2Needle
 - QwtCompassWindArrow, 133
- drawSymbols
 - QwtPlotCurve, 404
- drawText
 - QwtLegendItem, 263
 - QwtPainter, 308
 - QwtTextLabel, 767
- drawThermo
 - QwtThermo, 775
- drawThinNeedle
 - QwtCompassMagnetNeedle, 128
- drawThumb
 - QwtSlider, 719
- drawTick
 - QwtAbstractScaleDraw, 33
 - QwtDialScaleDraw, 186
 - QwtRoundScaleDraw, 657
 - QwtScaleDraw, 675
- drawTitle
 - QwtScaleWidget, 701
- drawTracker
 - QwtPicker, 328
 - QwtPlotPicker, 509
 - QwtPlotZoomer, 620
- drawTriangleNeedle
 - QwtCompassMagnetNeedle, 129
- drawWheel
 - QwtWheel, 790
- drawWheelBackground
 - QwtWheel, 790
- editable
 - QwtCounter, 138
- enableAxis
 - QwtPlot, 371
- enableComponent
 - QwtAbstractScaleDraw, 33
 - QwtDialScaleDraw, 186
 - QwtRoundScaleDraw, 657
 - QwtScaleDraw, 675
- enableX
 - QwtPlotGrid, 425
- enableXMin
 - QwtPlotGrid, 426
- enableY
 - QwtPlotGrid, 426
- enableYMin
 - QwtPlotGrid, 426
- end
 - QwtPicker, 328
 - QwtPlotPicker, 509
 - QwtPlotZoomer, 620
- endBorderDist
 - QwtScaleWidget, 701
- event
 - QwtCounter, 138
 - QwtPlot, 371
- eventFilter
 - QwtLegend, 255
 - QwtMagnifier, 293

- QwtPanner, [312](#)
- QwtPicker, [329](#)
- QwtPlotMagnifier, [461](#)
- QwtPlotPanner, [490](#)
- QwtPlotPicker, [509](#)
- QwtPlotRescaler, [549](#)
- QwtPlotZoomer, [620](#)
- exactPrevValue
 - QwtAbstractSlider, [41](#)
 - QwtAnalogClock, [64](#)
 - QwtCompass, [104](#)
 - QwtCounter, [138](#)
 - QwtDial, [161](#)
 - QwtDoubleRange, [206](#)
 - QwtKnob, [235](#)
 - QwtSlider, [719](#)
 - QwtWheel, [790](#)
- exactValue
 - QwtAbstractSlider, [41](#)
 - QwtAnalogClock, [64](#)
 - QwtCompass, [105](#)
 - QwtCounter, [138](#)
 - QwtDial, [162](#)
 - QwtDoubleRange, [206](#)
 - QwtKnob, [235](#)
 - QwtSlider, [719](#)
 - QwtWheel, [790](#)
- expandingDirection
 - QwtPlotRescaler, [549](#)
- expandingDirections
 - QwtDynGridLayout, [214](#)
- expandInterval
 - QwtPlotRescaler, [550](#)
- expandLineBreaks
 - QwtPlotLayout, [454](#)
- expandScale
 - QwtPlotRescaler, [550](#)
- extend
 - QwtDoubleInterval, [200](#)
- extent
 - QwtAbstractScaleDraw, [33](#)
 - QwtDialScaleDraw, [187](#)
 - QwtRoundScaleDraw, [658](#)
 - QwtScaleDraw, [675](#)
- fillBrush
 - QwtThermo, [775](#)
- fillColor
 - QwtThermo, [775](#)
- fillCurve
 - QwtPlotCurve, [404](#)
- fillRect
 - QwtPainter, [308](#)
- find
 - QwtLegend, [255](#)
- fitCurve
 - QwtCurveFitter, [151](#)
 - QwtSplineCurveFitter, [744](#)
- fitMode
 - QwtSplineCurveFitter, [744](#)
- fitValue
 - QwtAbstractSlider, [41](#)
 - QwtAnalogClock, [65](#)
 - QwtCompass, [105](#)
 - QwtCounter, [138](#)
 - QwtDial, [162](#)
 - QwtDoubleRange, [207](#)
 - QwtKnob, [235](#)
 - QwtSlider, [719](#)
 - QwtWheel, [791](#)
- floor125
 - QwtScaleArithmetic, [665](#)
- floorEps
 - QwtScaleArithmetic, [665](#)
- FocusIndicator
 - QwtPlotCanvas, [386](#)
- focusIndicator
 - QwtPlotCanvas, [388](#)
- font
 - QwtPlotPrintFilter, [530](#)
 - QwtPlotScaleItem, [560](#)
 - QwtText, [755](#)
- fontChange
 - QwtSlider, [720](#)
 - QwtThermo, [775](#)
- Format
 - QwtAlphaColorMap, [53](#)
 - QwtColorMap, [94](#)
 - QwtLinearColorMap, [273](#)
- format
 - QwtAlphaColorMap, [54](#)
 - QwtColorMap, [95](#)
 - QwtLinearColorMap, [276](#)
- frameShadow
 - QwtAnalogClock, [65](#)
 - QwtCompass, [105](#)
 - QwtDial, [162](#)
- getAbortKey
 - QwtPanner, [312](#)

- QwtPlotPanner, 490
- getBorderDistHint
 - QwtScaleDraw, 676
 - QwtScaleWidget, 702
- getMinBorderDist
 - QwtScaleWidget, 702
- getMouseButton
 - QwtMagnifier, 293
 - QwtPanner, 312
 - QwtPlotMagnifier, 462
 - QwtPlotPanner, 490
- getScrollMode
 - QwtAbstractSlider, 41
 - QwtAnalogClock, 65
 - QwtCompass, 105
 - QwtDial, 162
 - QwtSlider, 720
 - QwtWheel, 791
- getValue
 - QwtAbstractSlider, 42
 - QwtAnalogClock, 66
 - QwtCompass, 106
 - QwtDial, 163
 - QwtSlider, 720
 - QwtWheel, 791
- getZoomInKey
 - QwtMagnifier, 294
 - QwtPlotMagnifier, 462
- getZoomOutKey
 - QwtMagnifier, 294
 - QwtPlotMagnifier, 462
- grabProperties
 - QwtPlot, 372
- Hand
 - QwtAnalogClock, 60
- hand
 - QwtAnalogClock, 66
- hasComponent
 - QwtAbstractScaleDraw, 34
 - QwtDialScaleDraw, 187
 - QwtRoundScaleDraw, 658
 - QwtScaleDraw, 676
- hasHeightForWidth
 - QwtDynGridLayout, 214
- hasVisibleBackground
 - QwtAnalogClock, 67
 - QwtCompass, 106
 - QwtDial, 163
- heightForWidth
 - QwtDynGridLayout, 214
 - QwtLegend, 256
 - QwtLegendItem, 263
 - QwtMathMLTextEngine, 302
 - QwtPlainTextEngine, 357
 - QwtRichTextEngine, 652
 - QwtText, 755
 - QwtTextEngine, 764
 - QwtTextLabel, 768
- hide
 - QwtPlotCurve, 405
 - QwtPlotGrid, 427
 - QwtPlotItem, 442
 - QwtPlotMarker, 475
 - QwtPlotRasterItem, 537
 - QwtPlotScaleItem, 560
 - QwtPlotSpectrogram, 581
 - QwtPlotSvgItem, 598
- hideEvent
 - QwtPlotCanvas, 388
- horizontalScrollBar
 - QwtLegend, 256
- IdentifierMode
 - QwtLegendItem, 261
- identifierMode
 - QwtLegend, 256
 - QwtLegendItem, 263
- identifierWidth
 - QwtLegendItem, 263
- incPages
 - QwtAbstractSlider, 42
 - QwtAnalogClock, 67
 - QwtCompass, 106
 - QwtCounter, 139
 - QwtDial, 163
 - QwtDoubleRange, 207
 - QwtKnob, 235
 - QwtSlider, 720
 - QwtWheel, 792
- incSteps
 - QwtCounter, 139
- incValue
 - QwtAbstractSlider, 42
 - QwtAnalogClock, 67
 - QwtCompass, 107
 - QwtCounter, 139
 - QwtDial, 163
 - QwtDoubleRange, 207
 - QwtKnob, 236

- QwtSlider, [721](#)
- QwtWheel, [792](#)
- indent
 - QwtLegendItem, [263](#)
 - QwtTextLabel, [768](#)
- init
 - QwtPlotCurve, [405](#)
- initKeyPattern
 - QwtEventPattern, [223](#)
 - QwtPicker, [329](#)
 - QwtPlotPicker, [509](#)
 - QwtPlotZoomer, [621](#)
- initMousePattern
 - QwtEventPattern, [223](#)
 - QwtPicker, [329](#)
 - QwtPlotPicker, [510](#)
 - QwtPlotZoomer, [621](#)
- initRaster
 - QwtRasterData, [649](#)
- insert
 - QwtLegend, [256](#)
- insertLegend
 - QwtPlot, [372](#)
- internalBorder
 - QwtWheel, [792](#)
- intersect
 - QwtDoubleInterval, [200](#)
- intersects
 - QwtDoubleInterval, [200](#)
- interval
 - QwtIntervalData, [228](#)
 - QwtPlotRescaler, [550](#)
 - QwtScaleDiv, [668](#)
- invalidate
 - QwtDoubleInterval, [200](#)
 - QwtDynGridLayout, [215](#)
 - QwtPlotLayout, [454](#)
 - QwtScaleDiv, [668](#)
- invalidateCache
 - QwtAbstractScaleDraw, [34](#)
 - QwtDialScaleDraw, [187](#)
 - QwtPlotRasterItem, [537](#)
 - QwtPlotSpectrogram, [581](#)
 - QwtRoundScaleDraw, [658](#)
 - QwtScaleDraw, [676](#)
- invalidatePaintCache
 - QwtPlotCanvas, [388](#)
- invert
 - QwtScaleDiv, [668](#)
- inverted
 - QwtDoubleInterval, [200](#)
- invTransform
 - QwtPlot, [372](#)
 - QwtPlotCurve, [405](#)
 - QwtPlotGrid, [427](#)
 - QwtPlotItem, [442](#)
 - QwtPlotMarker, [475](#)
 - QwtPlotPicker, [510](#)
 - QwtPlotRasterItem, [537](#)
 - QwtPlotScaleItem, [560](#)
 - QwtPlotSpectrogram, [581](#)
 - QwtPlotSvgItem, [598](#)
 - QwtPlotZoomer, [621](#)
 - QwtScaleMap, [693](#)
- invXForm
 - QwtScaleTransformation, [697](#)
- isActive
 - QwtPicker, [330](#)
 - QwtPlotPicker, [510](#)
 - QwtPlotZoomer, [622](#)
- isAxisEnabled
 - QwtPlotMagnifier, [462](#)
 - QwtPlotPanner, [491](#)
- isChecked
 - QwtLegendItem, [264](#)
- isDown
 - QwtLegendItem, [264](#)
- isEmpty
 - QwtDynGridLayout, [215](#)
 - QwtLegend, [257](#)
 - QwtText, [756](#)
- isEnabled
 - QwtMagnifier, [294](#)
 - QwtPanner, [312](#)
 - QwtPicker, [330](#)
 - QwtPlotMagnifier, [463](#)
 - QwtPlotPanner, [491](#)
 - QwtPlotPicker, [510](#)
 - QwtPlotRescaler, [550](#)
 - QwtPlotZoomer, [622](#)
- isNull
 - QwtDoubleInterval, [201](#)
 - QwtText, [756](#)
- isOrientationEnabled
 - QwtPanner, [313](#)
 - QwtPlotPanner, [491](#)
- isReadOnly
 - QwtAbstractSlider, [43](#)
 - QwtAnalogClock, [67](#)
 - QwtCompass, [107](#)

- QwtDial, 164
- QwtKnob, 236
- QwtSlider, 721
- QwtWheel, 792
- isScaleDivFromAxis
 - QwtPlotScaleItem, 561
- isValid
 - QwtAbstractSlider, 43
 - QwtAnalogClock, 68
 - QwtCompass, 107
 - QwtCounter, 140
 - QwtDial, 164
 - QwtDoubleInterval, 201
 - QwtDoubleRange, 208
 - QwtKnob, 236
 - QwtScaleDiv, 668
 - QwtSlider, 721
 - QwtSpline, 741
 - QwtWheel, 793
- isVisible
 - QwtPlotCurve, 405
 - QwtPlotGrid, 427
 - QwtPlotItem, 443
 - QwtPlotMarker, 475
 - QwtPlotRasterItem, 537
 - QwtPlotScaleItem, 561
 - QwtPlotSpectrogram, 581
 - QwtPlotSvgItem, 598
- Item
 - QwtPlotPrintFilter, 529
- itemAt
 - QwtDynGridLayout, 215
- ItemAttribute
 - QwtPlotCurve, 396
 - QwtPlotGrid, 423
 - QwtPlotItem, 440
 - QwtPlotMarker, 472
 - QwtPlotRasterItem, 534
 - QwtPlotScaleItem, 557
 - QwtPlotSpectrogram, 575
 - QwtPlotSvgItem, 595
- itemChanged
 - QwtPlotCurve, 406
 - QwtPlotGrid, 427
 - QwtPlotItem, 443
 - QwtPlotMarker, 475
 - QwtPlotRasterItem, 538
 - QwtPlotScaleItem, 561
 - QwtPlotSpectrogram, 582
 - QwtPlotSvgItem, 598
- itemCount
 - QwtDynGridLayout, 215
 - QwtLegend, 257
- itemList
 - QwtPlot, 373
 - QwtPlotDict, 420
- itemMode
 - QwtLegend, 257
 - QwtLegendItem, 264
- keyFactor
 - QwtMagnifier, 294
 - QwtPlotMagnifier, 463
- keyMatch
 - QwtEventPattern, 224
 - QwtPicker, 330, 331
 - QwtPlotPicker, 511
 - QwtPlotZoomer, 622, 623
- keyPattern
 - QwtEventPattern, 225
 - QwtPicker, 331
 - QwtPlotPicker, 512
 - QwtPlotZoomer, 623
- KeyPatternCode
 - QwtEventPattern, 221
 - QwtPicker, 321
 - QwtPlotPicker, 500
 - QwtPlotZoomer, 612
- keyPressEvent
 - QwtAbstractSlider, 43
 - QwtAnalogClock, 68
 - QwtArrowButton, 90
 - QwtCompass, 107
 - QwtCounter, 140
 - QwtDial, 164
 - QwtKnob, 237
 - QwtLegendItem, 264
 - QwtSlider, 721
 - QwtWheel, 793
- keyReleaseEvent
 - QwtLegendItem, 264
- knobWidth
 - QwtKnob, 237
- label
 - QwtAbstractScaleDraw, 34
 - QwtDialScaleDraw, 188
 - QwtPlotMarker, 476
 - QwtRoundScaleDraw, 658
 - QwtScaleDraw, 676

- labelAlignment
 - QwtPlotMarker, 476
 - QwtScaleDraw, 677
- labelMap
 - QwtCompass, 108
- labelMatrix
 - QwtScaleDraw, 677
- labelOrientation
 - QwtPlotMarker, 476
- labelPosition
 - QwtScaleDraw, 677
- labelRect
 - QwtArrowButton, 90
 - QwtScaleDraw, 678
- labelRotation
 - QwtScaleDraw, 678
- labelSize
 - QwtScaleDraw, 678
- LayoutAttribute
 - QwtText, 753
- layoutContents
 - QwtLegend, 257
- layoutGrid
 - QwtDynGridLayout, 215
- layoutItems
 - QwtDynGridLayout, 216
- layoutLegend
 - QwtPlotLayout, 454
- layoutScale
 - QwtScaleWidget, 702
- layoutSlider
 - QwtSlider, 722
- layoutThermo
 - QwtThermo, 776
- layoutWheel
 - QwtWheel, 793
- legend
 - QwtPlot, 373
- legendChecked
 - QwtPlot, 373
- legendClicked
 - QwtPlot, 374
- LegendDisplayPolicy
 - QwtLegend, 253
- legendItem
 - QwtLegendItemManager, 271
 - QwtPlotCurve, 406
 - QwtPlotGrid, 427
 - QwtPlotItem, 443
 - QwtPlotMarker, 476
 - QwtPlotRasterItem, 538
 - QwtPlotScaleItem, 561
 - QwtPlotSpectrogram, 582
 - QwtPlotSvgItem, 599
- legendItemChecked
 - QwtPlot, 374
- legendItemClicked
 - QwtPlot, 374
- LegendItemMode
 - QwtLegend, 253
- legendItems
 - QwtLegend, 257
- LegendPosition
 - QwtPlot, 362
- legendPosition
 - QwtPlotLayout, 455
- legendRatio
 - QwtPlotLayout, 455
- legendRect
 - QwtPlotLayout, 455
- length
 - QwtScaleDraw, 678
- limited
 - QwtDoubleInterval, 201
- linePen
 - QwtPlotMarker, 477
- LineStyle
 - QwtPlotMarker, 472
- lineStyle
 - QwtPlotMarker, 477
- lineWidth
 - QwtAnalogClock, 69
 - QwtCompass, 108
 - QwtDial, 165
- loadData
 - QwtPlotSvgItem, 599
- loadFile
 - QwtPlotSvgItem, 599
- log10
 - QwtLog10ScaleEngine, 288
- lowerBound
 - QwtScaleDiv, 668
- lowerMargin
 - QwtLinearScaleEngine, 281
 - QwtLog10ScaleEngine, 288
 - QwtScaleEngine, 689
- majPen
 - QwtPlotGrid, 428
- majTickLength

- QwtAbstractScaleDraw, [34](#)
- QwtDialScaleDraw, [188](#)
- QwtRoundScaleDraw, [659](#)
- QwtScaleDraw, [678](#)
- map
 - QwtAbstractScaleDraw, [34](#)
 - QwtDialScaleDraw, [188](#)
 - QwtRoundScaleDraw, [659](#)
 - QwtScaleDraw, [679](#)
- margin
 - QwtLegendItem, [264](#)
 - QwtPlot, [374](#)
 - QwtPlotLayout, [455](#)
 - QwtScaleWidget, [702](#)
 - QwtTextLabel, [768](#)
- mass
 - QwtAbstractSlider, [44](#)
 - QwtAnalogClock, [69](#)
 - QwtCompass, [108](#)
 - QwtDial, [165](#)
 - QwtKnob, [237](#)
 - QwtSlider, [722](#)
 - QwtWheel, [793](#)
- maxCols
 - QwtDynGridLayout, [216](#)
- maxItemWidth
 - QwtDynGridLayout, [216](#)
- maxLabelHeight
 - QwtScaleDraw, [679](#)
- maxLabelWidth
 - QwtScaleDraw, [679](#)
- maxScaleArc
 - QwtAnalogClock, [69](#)
 - QwtCompass, [109](#)
 - QwtDial, [165](#)
- maxStackDepth
 - QwtPlotZoomer, [623](#)
- maxVal
 - QwtCounter, [140](#)
- maxValue
 - QwtAbstractSlider, [44](#)
 - QwtAnalogClock, [69](#)
 - QwtCompass, [109](#)
 - QwtCounter, [140](#)
 - QwtDial, [166](#)
 - QwtDoubleInterval, [201](#)
 - QwtDoubleRange, [208](#)
 - QwtKnob, [237](#)
 - QwtSlider, [722](#)
 - QwtThermo, [776](#)
 - QwtWheel, [794](#)
- maxXValue
 - QwtPlotCurve, [406](#)
- maxYValue
 - QwtPlotCurve, [406](#)
- metricsMap
 - QwtPainter, [308](#)
- mightRender
 - QwtMathMLTextEngine, [302](#)
 - QwtPlainTextEngine, [357](#)
 - QwtRichTextEngine, [652](#)
 - QwtTextEngine, [764](#)
- minimumExtent
 - QwtAbstractScaleDraw, [35](#)
 - QwtDialScaleDraw, [188](#)
 - QwtRoundScaleDraw, [659](#)
 - QwtScaleDraw, [679](#)
- minimumSizeHint
 - QwtAnalogClock, [69](#)
 - QwtArrowButton, [90](#)
 - QwtCompass, [109](#)
 - QwtDial, [166](#)
 - QwtKnob, [238](#)
 - QwtLegendItem, [264](#)
 - QwtPlot, [375](#)
 - QwtPlotLayout, [456](#)
 - QwtScaleWidget, [703](#)
 - QwtSlider, [723](#)
 - QwtTextLabel, [768](#)
 - QwtThermo, [776](#)
 - QwtWheel, [794](#)
- minLabelDist
 - QwtScaleDraw, [679](#)
- minLength
 - QwtScaleDraw, [680](#)
- minPen
 - QwtPlotGrid, [428](#)
- minScaleArc
 - QwtAnalogClock, [70](#)
 - QwtCompass, [109](#)
 - QwtDial, [166](#)
- minVal
 - QwtCounter, [140](#)
- minValue
 - QwtAbstractSlider, [44](#)
 - QwtAnalogClock, [70](#)
 - QwtCompass, [109](#)
 - QwtCounter, [141](#)
 - QwtDial, [166](#)
 - QwtDoubleInterval, [201](#)

- QwtDoubleRange, 208
- QwtKnob, 238
- QwtSlider, 723
- QwtThermo, 776
- QwtWheel, 794
- minXValue
 - QwtPlotCurve, 406
- minYValue
 - QwtPlotCurve, 406
- minZoomSize
 - QwtPlotZoomer, 624
- Mode
 - QwtAnalogClock, 60
 - QwtCompass, 100
 - QwtDial, 158
 - QwtLinearColorMap, 273
- mode
 - QwtAnalogClock, 70
 - QwtCompass, 110
 - QwtDial, 166
 - QwtLinearColorMap, 276
- mouseFactor
 - QwtMagnifier, 294
 - QwtPlotMagnifier, 463
- mouseMatch
 - QwtEventPattern, 225
 - QwtPicker, 331, 332
 - QwtPlotPicker, 512
 - QwtPlotZoomer, 624
- mouseMoveEvent
 - QwtAbstractSlider, 44
 - QwtAnalogClock, 70
 - QwtCompass, 110
 - QwtDial, 167
 - QwtKnob, 238
 - QwtSlider, 723
 - QwtWheel, 794
- mousePattern
 - QwtEventPattern, 226
 - QwtPicker, 332
 - QwtPlotPicker, 513
 - QwtPlotZoomer, 625
- MousePatternCode
 - QwtEventPattern, 221
 - QwtPicker, 322
 - QwtPlotPicker, 501
 - QwtPlotZoomer, 612
- mousePressEvent
 - QwtAbstractSlider, 45
 - QwtAnalogClock, 71
 - QwtCompass, 110
 - QwtDial, 167
 - QwtKnob, 238
 - QwtLegendItem, 265
 - QwtSlider, 723
 - QwtWheel, 795
- mouseReleaseEvent
 - QwtAbstractSlider, 45
 - QwtAnalogClock, 71
 - QwtCompass, 110
 - QwtDial, 167
 - QwtKnob, 239
 - QwtLegendItem, 265
 - QwtSlider, 724
 - QwtWheel, 795
- move
 - QwtPicker, 333
 - QwtPlotPicker, 513
 - QwtPlotZoomer, 625
 - QwtScaleDraw, 680
- moveBy
 - QwtPlotZoomer, 626
- moveCanvas
 - QwtPlotPanner, 491
- moveCenter
 - QwtDialScaleDraw, 188, 189
 - QwtRoundScaleDraw, 659, 660
- moved
 - QwtPanner, 313
 - QwtPicker, 333
 - QwtPlotPanner, 492
 - QwtPlotPicker, 513, 514
 - QwtPlotZoomer, 626
- needle
 - QwtAnalogClock, 71
 - QwtCompass, 111
 - QwtDial, 167, 168
- normalized
 - QwtDoubleInterval, 202
- num
 - QwtArrowButton, 90
- numButtons
 - QwtCounter, 141
- numCols
 - QwtDynGridLayout, 216
- numRows
 - QwtDynGridLayout, 217
- numThornLevels
 - QwtSimpleCompassRose, 711

- numThorns
 - QwtSimpleCompassRose, 711
- operator=
 - QwtAbstractScaleDraw, 35
 - QwtAlphaColorMap, 55
 - QwtArrayData, 86
 - QwtCPointerData, 148
 - QwtData, 153
 - QwtLinearColorMap, 276
 - QwtPolygonFData, 645
 - QwtRoundScaleDraw, 660
 - QwtScaleDraw, 681
 - QwtScaleMap, 694
 - QwtSpline, 741
 - QwtText, 756
- operator==
 - QwtDoubleInterval, 202
 - QwtScaleDiv, 669
 - QwtSymbol, 748
 - QwtText, 756
- operator&
 - QwtDoubleInterval, 202
- operator&=
 - QwtDoubleInterval, 202
- Options
 - QwtPlotLayout, 451
 - QwtPlotPrintFilter, 529
- options
 - QwtPlotPrintFilter, 530
- orientation
 - QwtAbstractSlider, 45
 - QwtAnalogClock, 71
 - QwtCompass, 111
 - QwtDial, 168
 - QwtKnob, 239
 - QwtPlotRescaler, 551
 - QwtScaleDraw, 681
 - QwtSlider, 724
 - QwtWheel, 795
- orientations
 - QwtPanner, 313
 - QwtPlotPanner, 492
- origin
 - QwtAnalogClock, 72
 - QwtCompass, 111
 - QwtDial, 168
- p1
 - QwtScaleMap, 694
- p2
 - QwtScaleMap, 694
- pageSize
 - QwtAbstractSlider, 45
 - QwtAnalogClock, 72
 - QwtCompass, 112
 - QwtCounter, 141
 - QwtDial, 168
 - QwtDoubleRange, 208
 - QwtKnob, 239
 - QwtSlider, 724
 - QwtWheel, 795
- PaintAttribute
 - QwtPlotCanvas, 386
 - QwtPlotCurve, 396
 - QwtText, 753
- paintCache
 - QwtPlotCanvas, 388, 389
- paintEvent
 - QwtAnalogClock, 72
 - QwtArrowButton, 90
 - QwtCompass, 112
 - QwtDial, 169
 - QwtKnob, 239
 - QwtLegendItem, 265
 - QwtPanner, 313
 - QwtPlotCanvas, 389
 - QwtPlotPanner, 492
 - QwtScaleWidget, 703
 - QwtSlider, 724
 - QwtTextLabel, 768
 - QwtThermo, 776
 - QwtWheel, 795
- paintRect
 - QwtPlotCurve, 407
 - QwtPlotGrid, 428
 - QwtPlotItem, 443
 - QwtPlotMarker, 477
 - QwtPlotRasterItem, 538
 - QwtPlotScaleItem, 561
 - QwtPlotSpectrogram, 582
 - QwtPlotSvgItem, 600
- palette
 - QwtCompassMagnetNeedle, 129
 - QwtCompassRose, 130
 - QwtCompassWindArrow, 134
 - QwtDialNeedle, 182
 - QwtDialSimpleNeedle, 196
 - QwtPlotScaleItem, 562
 - QwtSimpleCompassRose, 711

- paletteChange
 - QwtWheel, [795](#)
- panned
 - QwtPanner, [313](#)
 - QwtPlotPanner, [492](#)
- parentWidget
 - QwtMagnifier, [295](#)
 - QwtPicker, [333](#)
 - QwtPlotMagnifier, [463](#)
 - QwtPlotPicker, [514](#)
 - QwtPlotZoomer, [627](#)
- pDist
 - QwtScaleMap, [694](#)
- pen
 - QwtPlotCurve, [407](#)
 - QwtSymbol, [748](#)
- penWidth
 - QwtDialScaleDraw, [189](#)
 - QwtScaleWidget, [703](#)
- periodic
 - QwtAbstractSlider, [45](#)
 - QwtAnalogClock, [72](#)
 - QwtCompass, [112](#)
 - QwtCounter, [141](#)
 - QwtDial, [169](#)
 - QwtDoubleRange, [208](#)
 - QwtKnob, [239](#)
 - QwtSlider, [724](#)
 - QwtWheel, [796](#)
- pickRect
 - QwtPicker, [333](#)
 - QwtPlotPicker, [514](#)
 - QwtPlotZoomer, [627](#)
- pipeWidth
 - QwtThermo, [776](#)
- plot
 - QwtPlotCanvas, [389](#)
 - QwtPlotCurve, [407](#)
 - QwtPlotGrid, [429](#)
 - QwtPlotItem, [444](#)
 - QwtPlotMagnifier, [464](#)
 - QwtPlotMarker, [478](#)
 - QwtPlotPanner, [493](#)
 - QwtPlotPicker, [514](#), [515](#)
 - QwtPlotRasterItem, [539](#)
 - QwtPlotRescaler, [551](#)
 - QwtPlotScaleItem, [562](#)
 - QwtPlotSpectrogram, [583](#)
 - QwtPlotSvgItem, [600](#)
 - QwtPlotZoomer, [627](#)
- plotLayout
 - QwtPlot, [375](#)
- points
 - QwtSpline, [741](#)
- polish
 - QwtCounter, [141](#)
 - QwtPlot, [375](#)
- pos
 - QwtScaleDraw, [681](#)
- position
 - QwtPlotScaleItem, [562](#)
- pow10
 - QwtLog10ScaleEngine, [289](#)
- pressed
 - QwtLegendItem, [265](#)
- prevValue
 - QwtAbstractSlider, [46](#)
 - QwtAnalogClock, [73](#)
 - QwtCompass, [112](#)
 - QwtCounter, [141](#)
 - QwtDial, [169](#)
 - QwtDoubleRange, [209](#)
 - QwtKnob, [240](#)
 - QwtSlider, [724](#)
 - QwtWheel, [796](#)
- print
 - QwtPlot, [375](#), [376](#)
- printCanvas
 - QwtPlot, [376](#)
- printLegend
 - QwtPlot, [376](#)
- printLegendItem
 - QwtPlot, [376](#)
- printScale
 - QwtPlot, [377](#)
- printTitle
 - QwtPlot, [377](#)
- QwtAbstractScale, [23](#)
 - ~QwtAbstractScale, [24](#)
 - abstractScaleDraw, [25](#)
 - autoScale, [25](#)
 - QwtAbstractScale, [24](#)
 - rescale, [25](#)
 - scaleChange, [26](#)
 - scaleEngine, [26](#)
 - scaleMap, [26](#)
 - scaleMaxMajor, [26](#)
 - scaleMaxMinor, [27](#)
 - setAbstractScaleDraw, [27](#)

- setAutoScale, 27
- setScale, 27, 28
- setScaleEngine, 28
- setScaleMaxMajor, 28
- setScaleMaxMinor, 29
- QwtAbstractScaleDraw, 29
 - ~QwtAbstractScaleDraw, 32
 - draw, 32
 - drawBackbone, 32
 - drawLabel, 32
 - drawTick, 33
 - enableComponent, 33
 - extent, 33
 - hasComponent, 34
 - invalidateCache, 34
 - label, 34
 - majTickLength, 34
 - map, 34
 - minimumExtent, 35
 - operator=, 35
 - QwtAbstractScaleDraw, 31
 - ScaleComponent, 31
 - scaleDiv, 35
 - scaleMap, 35
 - setMinimumExtent, 35
 - setScaleDiv, 36
 - setSpacing, 36
 - setTickLength, 36
 - setTransformation, 36
 - spacing, 37
 - tickLabel, 37
 - tickLength, 37
- QwtAbstractSlider, 38
 - ~QwtAbstractSlider, 40
 - exactPrevValue, 41
 - exactValue, 41
 - fitValue, 41
 - getScrollMode, 41
 - getValue, 42
 - incPages, 42
 - incValue, 42
 - isReadOnly, 43
 - isValid, 43
 - keyPressEvent, 43
 - mass, 44
 - maxValue, 44
 - minValue, 44
 - mouseMoveEvent, 44
 - mousePressEvent, 45
 - mouseReleaseEvent, 45
 - orientation, 45
 - pageSize, 45
 - periodic, 45
 - prevValue, 46
 - QwtAbstractSlider, 40
 - rangeChange, 46
 - ScrollMode, 40
 - setMass, 46
 - setOrientation, 46
 - setPeriodic, 47
 - setPosition, 47
 - setRange, 47
 - setReadOnly, 48
 - setStep, 48
 - setTracking, 48
 - setUpdateTime, 49
 - setValid, 49
 - setValue, 49
 - sliderMoved, 49
 - sliderPressed, 50
 - sliderReleased, 50
 - step, 50
 - stepChange, 50
 - stopMoving, 50
 - timerEvent, 50
 - value, 51
 - valueChange, 51
 - valueChanged, 51
 - wheelEvent, 51
- QwtAlphaColorMap, 52
 - ~QwtAlphaColorMap, 53
 - color, 53, 54
 - colorTable, 54
 - copy, 54
 - Format, 53
 - format, 54
 - operator=, 55
 - QwtAlphaColorMap, 53
 - rgb, 55
 - setColor, 55
- QwtAnalogClock, 56
 - ~QwtAnalogClock, 61
 - boundingRect, 61
 - contentsRect, 61
 - Direction, 60
 - direction, 62
 - drawContents, 62
 - drawFocusIndicator, 62
 - drawFrame, 63
 - drawHand, 63

- drawNeedle, 63
- drawScale, 64
- drawScaleContents, 64
- exactPrevValue, 64
- exactValue, 64
- fitValue, 65
- frameShadow, 65
- getScrollMode, 65
- getValue, 66
- Hand, 60
- hand, 66
- hasVisibleBackground, 67
- incPages, 67
- incValue, 67
- isReadOnly, 67
- isValid, 68
- keyPressEvent, 68
- lineWidth, 69
- mass, 69
- maxScaleArc, 69
- maxValue, 69
- minimumSizeHint, 69
- minScaleArc, 70
- minValue, 70
- Mode, 60
- mode, 70
- mouseMoveEvent, 70
- mousePressEvent, 71
- mouseReleaseEvent, 71
- needle, 71
- orientation, 71
- origin, 72
- pageSize, 72
- paintEvent, 72
- periodic, 72
- prevValue, 73
- QwtAnalogClock, 61
- rangeChange, 73
- resizeEvent, 73
- scaleContentsRect, 73
- scaleDraw, 73
- scaleLabel, 73
- ScaleOptions, 60
- ScrollMode, 60
- setCurrentTime, 74
- setDirection, 74
- setFrameShadow, 74
- setHand, 74
- setLineWidth, 75
- setMass, 75
- setMode, 75
- setOrientation, 76
- setOrigin, 76
- setPeriodic, 76
- setPosition, 77
- setRange, 77
- setReadOnly, 77
- setScale, 78
- setScaleArc, 78
- setScaleDraw, 78
- setScaleOptions, 78
- setScaleTicks, 79
- setStep, 79
- setTime, 79
- setTracking, 80
- setUpdateTime, 80
- setValid, 80
- setValue, 81
- setWrapping, 81
- Shadow, 61
- showBackground, 81
- sizeHint, 82
- sliderMoved, 82
- sliderPressed, 82
- sliderReleased, 82
- step, 82
- stepChange, 83
- stopMoving, 83
- timerEvent, 83
- updateMask, 83
- updateScale, 83
- value, 83
- valueChange, 84
- valueChanged, 84
- wheelEvent, 84
- wrapping, 84
- QwtArrayData, 85
 - boundingRect, 86
 - copy, 86
 - operator=, 86
 - QwtArrayData, 86
 - size, 87
 - x, 87
 - xData, 87
 - y, 87
 - yData, 88
- QwtArrowButton, 88
 - ~QwtArrowButton, 89
 - arrowSize, 89
 - arrowType, 89

- drawArrow, [89](#)
- drawButtonLabel, [89](#)
- keyPressEvent, [90](#)
- labelRect, [90](#)
- minimumSizeHint, [90](#)
- num, [90](#)
- paintEvent, [90](#)
- QwtArrowButton, [88](#)
- sizeHint, [91](#)
- QwtClipper, [91](#)
 - clipCircle, [91](#)
 - clipPolygon, [92](#)
 - clipPolygonF, [92](#)
- QwtColorMap, [92](#)
 - ~QwtColorMap, [94](#)
 - color, [94](#)
 - colorIndex, [95](#)
 - colorTable, [95](#)
 - copy, [95](#)
 - Format, [94](#)
 - format, [95](#)
 - QwtColorMap, [94](#)
 - rgb, [96](#)
- QwtCompass, [96](#)
 - ~QwtCompass, [101](#)
 - boundingRect, [102](#)
 - contentsRect, [102](#)
 - Direction, [100](#)
 - direction, [102](#)
 - drawContents, [102](#)
 - drawFocusIndicator, [103](#)
 - drawFrame, [103](#)
 - drawNeedle, [103](#)
 - drawRose, [103](#)
 - drawScale, [104](#)
 - drawScaleContents, [104](#)
 - exactPrevValue, [104](#)
 - exactValue, [105](#)
 - fitValue, [105](#)
 - frameShadow, [105](#)
 - getScrollMode, [105](#)
 - getValue, [106](#)
 - hasVisibleBackground, [106](#)
 - incPages, [106](#)
 - incValue, [107](#)
 - isReadOnly, [107](#)
 - isValid, [107](#)
 - keyPressEvent, [107](#)
 - labelMap, [108](#)
 - lineWidth, [108](#)
 - mass, [108](#)
 - maxScaleArc, [109](#)
 - maxValue, [109](#)
 - minimumSizeHint, [109](#)
 - minScaleArc, [109](#)
 - minValue, [109](#)
 - Mode, [100](#)
 - mode, [110](#)
 - mouseMoveEvent, [110](#)
 - mousePressEvent, [110](#)
 - mouseReleaseEvent, [110](#)
 - needle, [111](#)
 - orientation, [111](#)
 - origin, [111](#)
 - pageSize, [112](#)
 - paintEvent, [112](#)
 - periodic, [112](#)
 - prevValue, [112](#)
 - QwtCompass, [101](#)
 - rangeChange, [112](#)
 - resizeEvent, [113](#)
 - rose, [113](#)
 - scaleContentsRect, [113](#)
 - scaleDraw, [113](#), [114](#)
 - scaleLabel, [114](#)
 - ScaleOptions, [100](#)
 - ScrollMode, [101](#)
 - setDirection, [114](#)
 - setFrameShadow, [114](#)
 - setLabelMap, [115](#)
 - setLineWidth, [115](#)
 - setMass, [115](#)
 - setMode, [116](#)
 - setNeedle, [116](#)
 - setOrientation, [116](#)
 - setOrigin, [117](#)
 - setPeriodic, [117](#)
 - setPosition, [117](#)
 - setRange, [118](#)
 - setReadOnly, [118](#)
 - setRose, [118](#)
 - setScale, [119](#)
 - setScaleArc, [119](#)
 - setScaleDraw, [119](#)
 - setScaleOptions, [119](#)
 - setScaleTicks, [120](#)
 - setStep, [120](#)
 - setTracking, [120](#)
 - setUpdateTime, [121](#)
 - setValid, [121](#)

- setValue, [121](#)
- setWrapping, [122](#)
- Shadow, [101](#)
- showBackground, [122](#)
- sizeHint, [123](#)
- sliderMoved, [123](#)
- sliderPressed, [123](#)
- sliderReleased, [123](#)
- step, [123](#)
- stepChange, [123](#)
- stopMoving, [124](#)
- timerEvent, [124](#)
- updateMask, [124](#)
- updateScale, [124](#)
- value, [124](#)
- valueChange, [125](#)
- valueChanged, [125](#)
- wheelEvent, [125](#)
- wrapping, [125](#)
- QwtCompassMagnetNeedle, [126](#)
 - draw, [128](#)
 - drawKnob, [128](#)
 - drawPointer, [128](#)
 - drawThinNeedle, [128](#)
 - drawTriangleNeedle, [129](#)
 - palette, [129](#)
 - QwtCompassMagnetNeedle, [127](#)
 - setPalette, [129](#)
 - Style, [127](#)
- QwtCompassRose, [129](#)
 - draw, [130](#)
 - palette, [130](#)
 - setPalette, [131](#)
- QwtCompassWindArrow, [131](#)
 - draw, [133](#)
 - drawKnob, [133](#)
 - drawStyle1Needle, [133](#)
 - drawStyle2Needle, [133](#)
 - palette, [134](#)
 - QwtCompassWindArrow, [132](#)
 - setPalette, [134](#)
 - Style, [132](#)
- QwtCounter, [134](#)
 - ~QwtCounter, [137](#)
 - Button, [137](#)
 - buttonReleased, [138](#)
 - editable, [138](#)
 - event, [138](#)
 - exactPrevValue, [138](#)
 - exactValue, [138](#)
 - fitValue, [138](#)
 - incPages, [139](#)
 - incSteps, [139](#)
 - incValue, [139](#)
 - isValid, [140](#)
 - keyPressEvent, [140](#)
 - maxVal, [140](#)
 - maxValue, [140](#)
 - minVal, [140](#)
 - minValue, [141](#)
 - numButtons, [141](#)
 - pageSize, [141](#)
 - periodic, [141](#)
 - polish, [141](#)
 - prevValue, [141](#)
 - QwtCounter, [137](#)
 - rangeChange, [142](#)
 - setEditable, [142](#)
 - setIncSteps, [142](#)
 - setMaxValue, [142](#)
 - setMinValue, [143](#)
 - setNumButtons, [143](#)
 - setPeriodic, [143](#)
 - setRange, [143](#)
 - setStep, [144](#)
 - setStepButton1, [144](#)
 - setStepButton2, [144](#)
 - setStepButton3, [144](#)
 - setValid, [145](#)
 - setValue, [145](#)
 - sizeHint, [145](#)
 - step, [145](#)
 - stepButton1, [145](#)
 - stepButton2, [145](#)
 - stepButton3, [146](#)
 - stepChange, [146](#)
 - value, [146](#)
 - valueChanged, [146](#)
 - wheelEvent, [146](#)
- QwtCPointerData, [147](#)
 - boundingRect, [148](#)
 - copy, [148](#)
 - operator=, [148](#)
 - QwtCPointerData, [147](#)
 - size, [148](#)
 - x, [148](#)
 - xData, [149](#)
 - y, [149](#)
 - yData, [149](#)
- QwtCurveFitter, [149](#)

- [~QwtCurveFitter](#), 150
 - [fitCurve](#), 151
 - [QwtCurveFitter](#), 150
- [QwtData](#), 151
 - [~QwtData](#), 152
 - [boundingRect](#), 152
 - [copy](#), 152
 - [operator=](#), 153
 - [QwtData](#), 152
 - [size](#), 153
 - [x](#), 153
 - [y](#), 153
- [QwtDial](#), 154
 - [~QwtDial](#), 159
 - [boundingRect](#), 159
 - [contentsRect](#), 159
 - [Direction](#), 158
 - [direction](#), 159
 - [drawContents](#), 159
 - [drawFocusIndicator](#), 160
 - [drawFrame](#), 160
 - [drawNeedle](#), 160
 - [drawScale](#), 161
 - [drawScaleContents](#), 161
 - [exactPrevValue](#), 161
 - [exactValue](#), 162
 - [fitValue](#), 162
 - [frameShadow](#), 162
 - [getScrollMode](#), 162
 - [getValue](#), 163
 - [hasVisibleBackground](#), 163
 - [incPages](#), 163
 - [incValue](#), 163
 - [isReadOnly](#), 164
 - [isValid](#), 164
 - [keyPressEvent](#), 164
 - [lineWidth](#), 165
 - [mass](#), 165
 - [maxScaleArc](#), 165
 - [maxValue](#), 166
 - [minimumSizeHint](#), 166
 - [minScaleArc](#), 166
 - [minValue](#), 166
 - [Mode](#), 158
 - [mode](#), 166
 - [mouseMoveEvent](#), 167
 - [mousePressEvent](#), 167
 - [mouseReleaseEvent](#), 167
 - [needle](#), 167, 168
 - [orientation](#), 168
 - [origin](#), 168
 - [pageSize](#), 168
 - [paintEvent](#), 169
 - [periodic](#), 169
 - [prevValue](#), 169
 - [QwtDial](#), 158
 - [rangeChange](#), 169
 - [resizeEvent](#), 169
 - [scaleContentsRect](#), 169
 - [scaleDraw](#), 170
 - [scaleLabel](#), 170
 - [ScaleOptions](#), 158
 - [ScrollMode](#), 158
 - [setDirection](#), 170
 - [setFrameShadow](#), 170
 - [setLineWidth](#), 171
 - [setMass](#), 171
 - [setMode](#), 171
 - [setNeedle](#), 172
 - [setOrientation](#), 172
 - [setOrigin](#), 172
 - [setPeriodic](#), 173
 - [setPosition](#), 173
 - [setRange](#), 173
 - [setReadOnly](#), 174
 - [setScale](#), 174
 - [setScaleArc](#), 174
 - [setScaleDraw](#), 174
 - [setScaleOptions](#), 175
 - [setScaleTicks](#), 175
 - [setStep](#), 175
 - [setTracking](#), 176
 - [setUpdateTime](#), 176
 - [setValid](#), 176
 - [setValue](#), 177
 - [setWrapping](#), 177
 - [Shadow](#), 158
 - [showBackground](#), 177
 - [sizeHint](#), 178
 - [sliderMoved](#), 178
 - [sliderPressed](#), 178
 - [sliderReleased](#), 178
 - [step](#), 178
 - [stepChange](#), 179
 - [stopMoving](#), 179
 - [timerEvent](#), 179
 - [updateMask](#), 179
 - [updateScale](#), 179
 - [value](#), 180
 - [valueChange](#), 180

- valueChanged, 180
- wheelEvent, 180
- wrapping, 180
- QwtDialNeedle, 181
 - ~QwtDialNeedle, 182
 - draw, 182
 - drawKnob, 182
 - palette, 182
 - QwtDialNeedle, 182
 - setPalette, 183
- QwtDialScaleDraw, 183
 - center, 185
 - draw, 185
 - drawBackbone, 185
 - drawLabel, 186
 - drawTick, 186
 - enableComponent, 186
 - extent, 187
 - hasComponent, 187
 - invalidateCache, 187
 - label, 188
 - majTickLength, 188
 - map, 188
 - minimumExtent, 188
 - moveCenter, 188, 189
 - penWidth, 189
 - QwtDialScaleDraw, 185
 - radius, 189
 - ScaleComponent, 185
 - scaleDiv, 189
 - scaleMap, 189
 - setAngleRange, 190
 - setMinimumExtent, 190
 - setPenWidth, 191
 - setRadius, 191
 - setScaleDiv, 191
 - setSpacing, 191
 - setTickLength, 192
 - setTransformation, 192
 - spacing, 192
 - tickLabel, 192
 - tickLength, 193
- QwtDialSimpleNeedle, 193
 - draw, 195
 - drawArrowNeedle, 195
 - drawKnob, 196
 - drawRayNeedle, 196
 - palette, 196
 - QwtDialSimpleNeedle, 195
 - setPalette, 196
 - setWidth, 196
 - Style, 194
 - width, 197
- QwtDoubleInterval, 197
 - borderFlags, 199
 - BorderMode, 198
 - contains, 199
 - extend, 200
 - intersect, 200
 - intersects, 200
 - invalidate, 200
 - inverted, 200
 - isNull, 201
 - isValid, 201
 - limited, 201
 - maxValue, 201
 - minValue, 201
 - normalized, 202
 - operator==, 202
 - operator&, 202
 - operator&=, 202
 - QwtDoubleInterval, 199
 - setBorderFlags, 203
 - setInterval, 203
 - setMaxValue, 204
 - setMinValue, 204
 - symmetrize, 204
 - unite, 204
 - width, 204
- QwtDoubleRange, 205
 - ~QwtDoubleRange, 206
 - exactPrevValue, 206
 - exactValue, 206
 - fitValue, 207
 - incPages, 207
 - incValue, 207
 - isValid, 208
 - maxValue, 208
 - minValue, 208
 - pageSize, 208
 - periodic, 208
 - prevValue, 209
 - QwtDoubleRange, 206
 - rangeChange, 209
 - setPeriodic, 209
 - setRange, 209
 - setStep, 210
 - setValid, 210
 - setValue, 210
 - step, 211

- stepChange, [211](#)
- value, [211](#)
- valueChange, [211](#)
- QwtDynGridLayout, [212](#)
 - ~QwtDynGridLayout, [213](#)
 - addItem, [213](#)
 - columnsForWidth, [213](#)
 - count, [214](#)
 - expandingDirections, [214](#)
 - hasHeightForWidth, [214](#)
 - heightForWidth, [214](#)
 - invalidate, [215](#)
 - isEmpty, [215](#)
 - itemAt, [215](#)
 - itemCount, [215](#)
 - layoutGrid, [215](#)
 - layoutItems, [216](#)
 - maxCols, [216](#)
 - maxItemWidth, [216](#)
 - numCols, [216](#)
 - numRows, [217](#)
 - QwtDynGridLayout, [213](#)
 - setExpandingDirections, [217](#)
 - setGeometry, [217](#)
 - setMaxCols, [218](#)
 - sizeHint, [218](#)
 - stretchGrid, [218](#)
 - takeAt, [218](#)
- QwtEventPattern, [219](#)
 - ~QwtEventPattern, [223](#)
 - initKeyPattern, [223](#)
 - initMousePattern, [223](#)
 - keyMatch, [224](#)
 - keyPattern, [225](#)
 - KeyPatternCode, [221](#)
 - mouseMatch, [225](#)
 - mousePattern, [226](#)
 - MousePatternCode, [221](#)
 - QwtEventPattern, [223](#)
 - setKeyPattern, [226](#)
 - setMousePattern, [227](#)
- QwtEventPattern::KeyPattern, [22](#)
- QwtEventPattern::MousePattern, [23](#)
- QwtIntervalData, [227](#)
 - ~QwtIntervalData, [228](#)
 - boundingRect, [228](#)
 - interval, [228](#)
 - QwtIntervalData, [228](#)
 - setData, [229](#)
 - size, [229](#)
 - value, [229](#)
- QwtKnob, [229](#)
 - ~QwtKnob, [233](#)
 - abstractScaleDraw, [233](#)
 - autoScale, [234](#)
 - borderWidth, [234](#)
 - draw, [234](#)
 - drawKnob, [234](#)
 - drawMarker, [234](#)
 - exactPrevValue, [235](#)
 - exactValue, [235](#)
 - fitValue, [235](#)
 - incPages, [235](#)
 - incValue, [236](#)
 - isReadOnly, [236](#)
 - isValid, [236](#)
 - keyPressEvent, [237](#)
 - knobWidth, [237](#)
 - mass, [237](#)
 - maxValue, [237](#)
 - minimumSizeHint, [238](#)
 - minValue, [238](#)
 - mouseMoveEvent, [238](#)
 - mousePressEvent, [238](#)
 - mouseReleaseEvent, [239](#)
 - orientation, [239](#)
 - pageSize, [239](#)
 - paintEvent, [239](#)
 - periodic, [239](#)
 - prevValue, [240](#)
 - QwtKnob, [233](#)
 - rescale, [240](#)
 - resizeEvent, [240](#)
 - scaleDraw, [240](#)
 - scaleEngine, [241](#)
 - scaleMap, [241](#)
 - scaleMaxMajor, [241](#)
 - scaleMaxMinor, [242](#)
 - ScrollMode, [233](#)
 - setAbstractScaleDraw, [242](#)
 - setAutoScale, [242](#)
 - setBorderWidth, [242](#)
 - setKnobWidth, [242](#)
 - setMass, [243](#)
 - setOrientation, [243](#)
 - setPeriodic, [243](#)
 - setPosition, [244](#)
 - setRange, [244](#)
 - setReadOnly, [244](#)
 - setScale, [245](#)

- setScaleDraw, [246](#)
- setScaleEngine, [246](#)
- setScaleMaxMajor, [246](#)
- setScaleMaxMinor, [246](#)
- setStep, [247](#)
- setSymbol, [247](#)
- setTotalAngle, [247](#)
- setTracking, [248](#)
- setUpdateTime, [248](#)
- setValid, [248](#)
- setValue, [249](#)
- sizeHint, [249](#)
- sliderMoved, [249](#)
- sliderPressed, [249](#)
- sliderReleased, [249](#)
- step, [250](#)
- stepChange, [250](#)
- stopMoving, [250](#)
- Symbol, [233](#)
- symbol, [250](#)
- timerEvent, [250](#)
- totalAngle, [251](#)
- value, [251](#)
- valueChanged, [251](#)
- wheelEvent, [251](#)
- QwtLegend, [251](#)
 - ~QwtLegend, [254](#)
 - clear, [254](#)
 - contentsWidget, [254](#)
 - displayPolicy, [255](#)
 - eventFilter, [255](#)
 - find, [255](#)
 - heightForWidth, [256](#)
 - horizontalScrollBar, [256](#)
 - identifierMode, [256](#)
 - insert, [256](#)
 - isEmpty, [257](#)
 - itemCount, [257](#)
 - itemMode, [257](#)
 - layoutContents, [257](#)
 - LegendDisplayPolicy, [253](#)
 - LegendItemMode, [253](#)
 - legendItems, [257](#)
 - QwtLegend, [254](#)
 - remove, [257](#)
 - resizeEvent, [257](#)
 - setDisplayPolicy, [258](#)
 - setItemMode, [258](#)
 - sizeHint, [258](#)
 - verticalScrollBar, [258](#)
- QwtLegendItem, [259](#)
 - ~QwtLegendItem, [261](#)
 - checked, [262](#)
 - clear, [262](#)
 - clicked, [262](#)
 - curvePen, [262](#)
 - drawContents, [262](#)
 - drawIdentifier, [262](#)
 - drawItem, [263](#)
 - drawText, [263](#)
 - heightForWidth, [263](#)
 - IdentifierMode, [261](#)
 - identifierMode, [263](#)
 - identifierWidth, [263](#)
 - indent, [263](#)
 - isChecked, [264](#)
 - isDown, [264](#)
 - itemMode, [264](#)
 - keyPressEvent, [264](#)
 - keyReleaseEvent, [264](#)
 - margin, [264](#)
 - minimumSizeHint, [264](#)
 - mousePressEvent, [265](#)
 - mouseReleaseEvent, [265](#)
 - paintEvent, [265](#)
 - pressed, [265](#)
 - QwtLegendItem, [261](#)
 - released, [265](#)
 - setChecked, [265](#)
 - setCurvePen, [266](#)
 - setDown, [266](#)
 - setIdentifierMode, [266](#)
 - setIdentifierWidth, [266](#)
 - setIndent, [266](#)
 - setItemMode, [267](#)
 - setMargin, [267](#)
 - setSpacing, [267](#)
 - setSymbol, [267](#)
 - setText, [268](#)
 - sizeHint, [268](#)
 - spacing, [268](#)
 - symbol, [268](#)
 - text, [269](#)
 - textRect, [269](#)
- QwtLegendItemManager, [269](#)
 - ~QwtLegendItemManager, [270](#)
 - legendItem, [271](#)
 - QwtLegendItemManager, [270](#)
 - updateLegend, [271](#)
- QwtLinearColorMap, [271](#)

- [~QwtLinearColorMap](#), 274
 - [addColorStop](#), 274
 - [color](#), 274
 - [color1](#), 275
 - [color2](#), 275
 - [colorIndex](#), 275
 - [colorStops](#), 275
 - [colorTable](#), 275
 - [copy](#), 276
 - [Format](#), 273
 - [format](#), 276
 - [Mode](#), 273
 - [mode](#), 276
 - [operator=](#), 276
 - [QwtLinearColorMap](#), 273, 274
 - [rgb](#), 277
 - [setColorInterval](#), 277
 - [setMode](#), 277
- [QwtLinearScaleEngine](#), 278
 - [align](#), 279
 - [Attribute](#), 279
 - [attributes](#), 280
 - [autoScale](#), 280
 - [buildInterval](#), 280
 - [contains](#), 280
 - [divideInterval](#), 281
 - [divideScale](#), 281
 - [lowerMargin](#), 281
 - [reference](#), 282
 - [setAttribute](#), 282
 - [setAttributes](#), 282
 - [setMargins](#), 282
 - [setReference](#), 283
 - [strip](#), 283
 - [testAttribute](#), 284
 - [transformation](#), 284
 - [upperMargin](#), 284
- [QwtLog10ScaleEngine](#), 284
 - [Attribute](#), 286
 - [attributes](#), 287
 - [autoScale](#), 287
 - [buildInterval](#), 287
 - [contains](#), 287
 - [divideInterval](#), 288
 - [divideScale](#), 288
 - [log10](#), 288
 - [lowerMargin](#), 288
 - [pow10](#), 289
 - [reference](#), 289
 - [setAttribute](#), 289
 - [setAttributes](#), 289
 - [setMargins](#), 290
 - [setReference](#), 290
 - [strip](#), 290
 - [testAttribute](#), 291
 - [transformation](#), 291
 - [upperMargin](#), 291
- [QwtMagnifier](#), 291
 - [~QwtMagnifier](#), 293
 - [eventFilter](#), 293
 - [getMouseButton](#), 293
 - [getZoomInKey](#), 294
 - [getZoomOutKey](#), 294
 - [isEnabled](#), 294
 - [keyFactor](#), 294
 - [mouseFactor](#), 294
 - [parentWidget](#), 295
 - [QwtMagnifier](#), 293
 - [rescale](#), 295
 - [setEnabled](#), 295
 - [setKeyFactor](#), 296
 - [setMouseButton](#), 296
 - [setMouseFactor](#), 296
 - [setWheelButtonState](#), 297
 - [setWheelFactor](#), 297
 - [setZoomInKey](#), 297
 - [setZoomOutKey](#), 298
 - [wheelButtonState](#), 298
 - [wheelFactor](#), 298
 - [widgetKeyPressEvent](#), 298
 - [widgetKeyReleaseEvent](#), 299
 - [widgetMouseMoveEvent](#), 299
 - [widgetMousePressEvent](#), 299
 - [widgetMouseReleaseEvent](#), 299
 - [widgetWheelEvent](#), 300
- [QwtMathMLTextEngine](#), 300
 - [~QwtMathMLTextEngine](#), 301
 - [draw](#), 302
 - [heightForWidth](#), 302
 - [mightRender](#), 302
 - [QwtMathMLTextEngine](#), 301
 - [textMargins](#), 302
 - [textSize](#), 303
- [QwtMetricsMap](#), 303
 - [translate](#), 304, 305
- [QwtPainter](#), 305
 - [deviceClipping](#), 306
 - [deviceClipRect](#), 306
 - [drawEllipse](#), 306
 - [drawLine](#), 307

- drawPie, 307
- drawPoint, 307
- drawPolygon, 307
- drawPolyline, 307
- drawRect, 307
- drawRoundFrame, 308
- drawSimpleRichText, 308
- drawText, 308
- fillRect, 308
- metricsMap, 308
- resetMetricsMap, 309
- scaledPen, 309
- setClipRect, 309
- setDeviceClipping, 309
- setMetricsMap, 309, 310
- QwtPanner, 310
 - ~QwtPanner, 312
 - cursor, 312
 - eventFilter, 312
 - getAbortKey, 312
 - getMouseButton, 312
 - isEnabled, 312
 - isOrientationEnabled, 313
 - moved, 313
 - orientations, 313
 - paintEvent, 313
 - panned, 313
 - QwtPanner, 311
 - setAbortKey, 314
 - setCursor, 314
 - setEnabled, 314
 - setMouseButton, 314
 - setOrientations, 315
 - widgetKeyPressEvent, 315
 - widgetKeyReleaseEvent, 315
 - widgetMouseMoveEvent, 315
 - widgetMousePressEvent, 315
 - widgetMouseReleaseEvent, 316
- QwtPicker, 316
 - ~QwtPicker, 326
 - accept, 327
 - append, 327
 - appended, 327
 - begin, 327
 - changed, 328
 - DisplayMode, 321
 - drawRubberBand, 328
 - drawTracker, 328
 - end, 328
 - eventFilter, 329
 - initKeyPattern, 329
 - initMousePattern, 329
 - isActive, 330
 - isEnabled, 330
 - keyMatch, 330, 331
 - keyPattern, 331
 - KeyPatternCode, 321
 - mouseMatch, 331, 332
 - mousePattern, 332
 - MousePatternCode, 322
 - move, 333
 - moved, 333
 - parentWidget, 333
 - pickRect, 333
 - QwtPicker, 326
 - RectSelectionType, 323
 - reset, 333
 - SizeMode, 324
 - resizeMode, 334
 - RubberBand, 324
 - rubberBand, 334
 - rubberBandPen, 334
 - rubberBandWidget, 334
 - selected, 334
 - selection, 335
 - selectionFlags, 335
 - SelectionMode, 325
 - SelectionType, 325
 - setEnabled, 335
 - setKeyPattern, 335
 - setMousePattern, 336
 - setSizeMode, 336
 - setRubberBand, 337
 - setRubberBandPen, 337
 - setSelectionFlags, 337
 - setTrackerFont, 337
 - setTrackerMode, 338
 - setTrackerPen, 338
 - stateMachine, 338
 - stretchSelection, 339
 - trackerFont, 339
 - trackerMode, 340
 - trackerPen, 340
 - trackerPosition, 340
 - trackerRect, 340
 - trackerText, 340
 - trackerWidget, 341
 - transition, 341
 - updateDisplay, 341
 - widgetKeyPressEvent, 341

- [widgetKeyReleaseEvent](#), 342
 - [widgetLeaveEvent](#), 342
 - [widgetMouseDoubleClickEvent](#), 342
 - [widgetMouseMoveEvent](#), 343
 - [widgetMousePressEvent](#), 343
 - [widgetMouseReleaseEvent](#), 343
 - [widgetWheelEvent](#), 343
- [QwtPickerClickPointMachine](#), 344
 - [Command](#), 345
 - [reset](#), 345
 - [setState](#), 345
 - [state](#), 345
 - [transition](#), 345
- [QwtPickerClickRectMachine](#), 346
 - [Command](#), 347
 - [reset](#), 347
 - [setState](#), 347
 - [state](#), 347
 - [transition](#), 347
- [QwtPickerDragPointMachine](#), 347
 - [Command](#), 348
 - [reset](#), 349
 - [setState](#), 349
 - [state](#), 349
 - [transition](#), 349
- [QwtPickerDragRectMachine](#), 349
 - [Command](#), 351
 - [reset](#), 351
 - [setState](#), 351
 - [state](#), 351
 - [transition](#), 351
- [QwtPickerMachine](#), 351
 - [~QwtPickerMachine](#), 353
 - [Command](#), 353
 - [QwtPickerMachine](#), 353
 - [reset](#), 353
 - [setState](#), 353
 - [state](#), 353
 - [transition](#), 353
- [QwtPickerPolygonMachine](#), 354
 - [Command](#), 355
 - [reset](#), 355
 - [setState](#), 355
 - [state](#), 355
 - [transition](#), 355
- [QwtPlainTextEngine](#), 356
 - [~QwtPlainTextEngine](#), 356
 - [draw](#), 357
 - [heightForWidth](#), 357
 - [mightRender](#), 357
 - [QwtPlainTextEngine](#), 356
 - [textMargins](#), 358
 - [textSize](#), 358
- [QwtPlot](#), 358
 - [~QwtPlot](#), 364
 - [applyProperties](#), 364
 - [autoDelete](#), 364
 - [autoRefresh](#), 364
 - [autoReplot](#), 364
 - [Axis](#), 362
 - [axisAutoScale](#), 364
 - [axisEnabled](#), 365
 - [axisFont](#), 365
 - [axisMaxMajor](#), 365
 - [axisMaxMinor](#), 365
 - [axisScaleDiv](#), 366
 - [axisScaleDraw](#), 366, 367
 - [axisScaleEngine](#), 367, 368
 - [axisStepSize](#), 368
 - [axisTitle](#), 368
 - [axisValid](#), 368
 - [axisWidget](#), 369
 - [canvas](#), 369
 - [canvasBackground](#), 369
 - [canvasLineWidth](#), 370
 - [canvasMap](#), 370
 - [clear](#), 370
 - [detachItems](#), 370
 - [drawCanvas](#), 371
 - [drawItems](#), 371
 - [enableAxis](#), 371
 - [event](#), 371
 - [grabProperties](#), 372
 - [insertLegend](#), 372
 - [invTransform](#), 372
 - [itemList](#), 373
 - [legend](#), 373
 - [legendChecked](#), 373
 - [legendClicked](#), 374
 - [legendItemChecked](#), 374
 - [legendItemClicked](#), 374
 - [LegendPosition](#), 362
 - [margin](#), 374
 - [minimumSizeHint](#), 375
 - [plotLayout](#), 375
 - [polish](#), 375
 - [print](#), 375, 376
 - [printCanvas](#), 376
 - [printLegend](#), 376
 - [printLegendItem](#), 376

- printScale, [377](#)
- printTitle, [377](#)
- QwtPlot, [363](#)
- replot, [377](#)
- resizeEvent, [378](#)
- setAutoDelete, [378](#)
- setAutoReplot, [378](#)
- setAxisAutoScale, [378](#)
- setAxisFont, [379](#)
- setAxisLabelAlignment, [379](#)
- setAxisLabelRotation, [379](#)
- setAxisMaxMajor, [380](#)
- setAxisMaxMinor, [380](#)
- setAxisScale, [380](#)
- setAxisScaleDiv, [381](#)
- setAxisScaleDraw, [381](#)
- setAxisScaleEngine, [381](#)
- setAxisTitle, [382](#)
- setCanvasBackground, [382](#)
- setCanvasLineWidth, [382](#)
- setMargin, [383](#)
- setTitle, [383](#)
- sizeHint, [383](#)
- title, [383](#)
- titleLabel, [384](#)
- transform, [384](#)
- updateAxes, [384](#)
- updateLayout, [384](#)
- updateTabOrder, [385](#)
- QwtPlotCanvas, [385](#)
 - ~QwtPlotCanvas, [387](#)
 - drawCanvas, [387](#)
 - drawContents, [387](#)
 - drawFocusIndicator, [388](#)
 - FocusIndicator, [386](#)
 - focusIndicator, [388](#)
 - hideEvent, [388](#)
 - invalidatePaintCache, [388](#)
 - PaintAttribute, [386](#)
 - paintCache, [388](#), [389](#)
 - paintEvent, [389](#)
 - plot, [389](#)
 - QwtPlotCanvas, [387](#)
 - replot, [389](#)
 - setFocusIndicator, [389](#)
 - setPaintAttribute, [389](#)
 - testPaintAttribute, [390](#)
- QwtPlotCurve, [390](#)
 - ~QwtPlotCurve, [397](#)
 - attach, [398](#)
 - baseline, [398](#)
 - boundingRect, [398](#)
 - brush, [398](#)
 - closePolyline, [399](#)
 - closestPoint, [399](#)
 - CurveAttribute, [395](#)
 - curveFitter, [399](#)
 - CurveStyle, [395](#)
 - CurveType, [395](#)
 - curveType, [399](#)
 - data, [400](#)
 - dataSize, [400](#)
 - detach, [400](#)
 - draw, [400](#), [401](#)
 - drawCurve, [402](#)
 - drawDots, [402](#)
 - drawLines, [403](#)
 - drawSteps, [403](#)
 - drawSticks, [403](#)
 - drawSymbols, [404](#)
 - fillCurve, [404](#)
 - hide, [405](#)
 - init, [405](#)
 - invTransform, [405](#)
 - isVisible, [405](#)
 - ItemAttribute, [396](#)
 - itemChanged, [406](#)
 - legendItem, [406](#)
 - maxXValue, [406](#)
 - maxYValue, [406](#)
 - minXValue, [406](#)
 - minYValue, [406](#)
 - PaintAttribute, [396](#)
 - paintRect, [407](#)
 - pen, [407](#)
 - plot, [407](#)
 - QwtPlotCurve, [397](#)
 - RenderHint, [397](#)
 - rtti, [407](#)
 - RttiValues, [397](#)
 - scaleRect, [407](#)
 - setAxis, [408](#)
 - setBaseline, [408](#)
 - setBrush, [408](#)
 - setCurveAttribute, [409](#)
 - setCurveFitter, [409](#)
 - setCurveType, [409](#)
 - setData, [410](#)
 - setItemAttribute, [411](#)
 - setPaintAttribute, [411](#)

- setPen, [411](#)
- setRawData, [411](#)
- setRenderHint, [412](#)
- setStyle, [412](#)
- setSymbol, [412](#)
- setTitle, [413](#)
- setVisible, [413](#)
- setXAxis, [414](#)
- setYAxis, [414](#)
- setZ, [414](#)
- show, [414](#)
- style, [415](#)
- symbol, [415](#)
- testCurveAttribute, [415](#)
- testItemAttribute, [415](#)
- testPaintAttribute, [416](#)
- testRenderHint, [416](#)
- title, [416](#)
- transform, [416](#)
- updateLegend, [417](#)
- updateScaleDiv, [417](#)
- x, [417](#)
- xAxis, [418](#)
- y, [418](#)
- yAxis, [418](#)
- z, [418](#)
- QwtPlotDict, [418](#)
 - ~QwtPlotDict, [420](#)
 - autoDelete, [420](#)
 - detachItems, [420](#)
 - itemList, [420](#)
 - QwtPlotDict, [419](#)
 - setAutoDelete, [421](#)
- QwtPlotGrid, [421](#)
 - ~QwtPlotGrid, [424](#)
 - attach, [424](#)
 - boundingRect, [425](#)
 - detach, [425](#)
 - draw, [425](#)
 - enableX, [425](#)
 - enableXMin, [426](#)
 - enableY, [426](#)
 - enableYMin, [426](#)
 - hide, [427](#)
 - invTransform, [427](#)
 - isVisible, [427](#)
 - ItemAttribute, [423](#)
 - itemChanged, [427](#)
 - legendItem, [427](#)
 - majPen, [428](#)
 - minPen, [428](#)
 - paintRect, [428](#)
 - plot, [429](#)
 - QwtPlotGrid, [424](#)
 - RenderHint, [424](#)
 - rtti, [429](#)
 - RttiValues, [424](#)
 - scaleRect, [429](#)
 - setAxis, [429](#)
 - setItemAttribute, [430](#)
 - setMajPen, [430](#)
 - setMinPen, [430](#)
 - setPen, [430](#)
 - setRenderHint, [431](#)
 - setTitle, [431](#)
 - setVisible, [431](#)
 - setXAxis, [432](#)
 - setXDiv, [432](#)
 - setYAxis, [432](#)
 - setYDiv, [432](#)
 - setZ, [433](#)
 - show, [433](#)
 - testItemAttribute, [433](#)
 - testRenderHint, [433](#)
 - title, [434](#)
 - transform, [434](#)
 - updateLegend, [434](#)
 - updateScaleDiv, [435](#)
 - xAxis, [435](#)
 - xEnabled, [435](#)
 - xMinEnabled, [436](#)
 - xScaleDiv, [436](#)
 - yAxis, [436](#)
 - yEnabled, [436](#)
 - yMinEnabled, [436](#)
 - yScaleDiv, [437](#)
 - z, [437](#)
- QwtPlotItem, [437](#)
 - ~QwtPlotItem, [441](#)
 - attach, [441](#)
 - boundingRect, [441](#)
 - detach, [442](#)
 - draw, [442](#)
 - hide, [442](#)
 - invTransform, [442](#)
 - isVisible, [443](#)
 - ItemAttribute, [440](#)
 - itemChanged, [443](#)
 - legendItem, [443](#)
 - paintRect, [443](#)

- plot, [444](#)
- QwtPlotItem, [441](#)
- RenderHint, [440](#)
- rtti, [444](#)
- RttiValues, [440](#)
- scaleRect, [444](#)
- setAxis, [445](#)
- setItemAttribute, [445](#)
- setRenderHint, [445](#)
- setTitle, [446](#)
- setVisible, [446](#)
- setXAxis, [446](#)
- setYAxis, [447](#)
- setZ, [447](#)
- show, [447](#)
- testItemAttribute, [447](#)
- testRenderHint, [448](#)
- title, [448](#)
- transform, [448](#)
- updateLegend, [449](#)
- updateScaleDiv, [449](#)
- xAxis, [449](#)
- yAxis, [450](#)
- z, [450](#)
- QwtPlotLayout, [450](#)
 - ~QwtPlotLayout, [452](#)
 - activate, [452](#)
 - alignCanvasToScales, [452](#)
 - alignLegend, [453](#)
 - alignScales, [453](#)
 - canvasMargin, [453](#)
 - canvasRect, [453](#)
 - expandLineBreaks, [454](#)
 - invalidate, [454](#)
 - layoutLegend, [454](#)
 - legendPosition, [455](#)
 - legendRatio, [455](#)
 - legendRect, [455](#)
 - margin, [455](#)
 - minimumSizeHint, [456](#)
 - Options, [451](#)
 - QwtPlotLayout, [452](#)
 - scaleRect, [456](#)
 - setAlignCanvasToScales, [456](#)
 - setCanvasMargin, [457](#)
 - setLegendPosition, [457](#)
 - setLegendRatio, [458](#)
 - setMargin, [458](#)
 - setSpacing, [458](#)
 - spacing, [459](#)
 - titleRect, [459](#)
- QwtPlotMagnifier, [459](#)
 - ~QwtPlotMagnifier, [461](#)
 - canvas, [461](#)
 - eventFilter, [461](#)
 - getMouseButton, [462](#)
 - getZoomInKey, [462](#)
 - getZoomOutKey, [462](#)
 - isAxisEnabled, [462](#)
 - isEnabled, [463](#)
 - keyFactor, [463](#)
 - mouseFactor, [463](#)
 - parentWidget, [463](#)
 - plot, [464](#)
 - QwtPlotMagnifier, [461](#)
 - rescale, [464](#)
 - setAxisEnabled, [464](#)
 - setEnabled, [464](#)
 - setKeyFactor, [465](#)
 - setMouseButton, [465](#)
 - setMouseFactor, [465](#)
 - setWheelButtonState, [466](#)
 - setWheelFactor, [466](#)
 - setZoomInKey, [466](#)
 - setZoomOutKey, [467](#)
 - wheelButtonState, [467](#)
 - wheelFactor, [467](#)
 - widgetKeyPressEvent, [468](#)
 - widgetKeyReleaseEvent, [468](#)
 - widgetMouseMoveEvent, [468](#)
 - widgetMousePressEvent, [468](#)
 - widgetMouseReleaseEvent, [469](#)
 - widgetWheelEvent, [469](#)
- QwtPlotMarker, [469](#)
 - ~QwtPlotMarker, [473](#)
 - attach, [473](#)
 - boundingRect, [474](#)
 - detach, [474](#)
 - draw, [474](#)
 - drawAt, [474](#)
 - hide, [475](#)
 - invTransform, [475](#)
 - isVisible, [475](#)
 - ItemAttribute, [472](#)
 - itemChanged, [475](#)
 - label, [476](#)
 - labelAlignment, [476](#)
 - labelOrientation, [476](#)
 - legendItem, [476](#)
 - linePen, [477](#)

- LineStyle, [472](#)
- lineStyle, [477](#)
- paintRect, [477](#)
- plot, [478](#)
- QwtPlotMarker, [473](#)
- RenderHint, [473](#)
- rtti, [478](#)
- RttiValues, [473](#)
- scaleRect, [478](#)
- setAxis, [478](#)
- setItemAttribute, [479](#)
- setLabel, [479](#)
- setLabelAlignment, [479](#)
- setLabelOrientation, [480](#)
- setLinePen, [480](#)
- setLineStyle, [480](#)
- setRenderHint, [480](#)
- setSpacing, [481](#)
- setSymbol, [481](#)
- setTitle, [481](#), [482](#)
- setValue, [482](#)
- setVisible, [482](#)
- setXAxis, [482](#)
- setXValue, [483](#)
- setYAxis, [483](#)
- setYValue, [483](#)
- setZ, [483](#)
- show, [484](#)
- spacing, [484](#)
- symbol, [484](#)
- testItemAttribute, [484](#)
- testRenderHint, [484](#)
- title, [485](#)
- transform, [485](#)
- updateLegend, [485](#)
- updateScaleDiv, [486](#)
- value, [486](#)
- xAxis, [486](#)
- xValue, [487](#)
- yAxis, [487](#)
- yValue, [487](#)
- z, [487](#)
- QwtPlotPanner, [487](#)
 - ~QwtPlotPanner, [489](#)
 - canvas, [490](#)
 - cursor, [490](#)
 - eventFilter, [490](#)
 - getAbortKey, [490](#)
 - getMouseButton, [490](#)
 - isEnabled, [491](#)
 - isOrientationEnabled, [491](#)
 - moveCanvas, [491](#)
 - moved, [492](#)
 - orientations, [492](#)
 - paintEvent, [492](#)
 - panned, [492](#)
 - plot, [493](#)
 - QwtPlotPanner, [489](#)
 - setAbortKey, [493](#)
 - setAxisEnabled, [493](#)
 - setCursor, [493](#)
 - setEnabled, [494](#)
 - setMouseButton, [494](#)
 - setOrientations, [494](#)
 - widgetKeyPressEvent, [494](#)
 - widgetKeyReleaseEvent, [494](#)
 - widgetMouseMoveEvent, [495](#)
 - widgetMousePressEvent, [495](#)
 - widgetMouseReleaseEvent, [495](#)
- QwtPlotPicker, [496](#)
 - ~QwtPlotPicker, [505](#)
 - accept, [506](#)
 - append, [506](#)
 - appended, [507](#)
 - begin, [507](#)
 - canvas, [508](#)
 - changed, [508](#)
 - DisplayMode, [500](#)
 - drawRubberBand, [508](#)
 - drawTracker, [509](#)
 - end, [509](#)
 - eventFilter, [509](#)
 - initKeyPattern, [509](#)
 - initMousePattern, [510](#)
 - invTransform, [510](#)
 - isActive, [510](#)
 - isEnabled, [510](#)
 - keyMatch, [511](#)
 - keyPattern, [512](#)
 - KeyPatternCode, [500](#)
 - mouseMatch, [512](#)
 - mousePattern, [513](#)
 - MousePatternCode, [501](#)
 - move, [513](#)
 - moved, [513](#), [514](#)
 - parentWidget, [514](#)
 - pickRect, [514](#)
 - plot, [514](#), [515](#)
 - QwtPlotPicker, [505](#), [506](#)

- RectSelectionType, [502](#)
- reset, [515](#)
- SizeMode, [503](#)
- resizeMode, [515](#)
- RubberBand, [503](#)
- rubberBand, [515](#)
- rubberBandPen, [515](#)
- rubberBandWidget, [516](#)
- scaleRect, [516](#)
- selected, [516](#), [517](#)
- selection, [517](#)
- selectionFlags, [517](#)
- SelectionMode, [504](#)
- SelectionType, [504](#)
- setAxis, [517](#)
- setEnabled, [517](#)
- setKeyPattern, [518](#)
- setMousePattern, [518](#)
- setSizeMode, [519](#)
- setRubberBand, [519](#)
- setRubberBandPen, [519](#)
- setSelectionFlags, [520](#)
- setTrackerFont, [520](#)
- setTrackerMode, [520](#)
- setTrackerPen, [521](#)
- stateMachine, [521](#)
- stretchSelection, [522](#)
- trackerFont, [522](#)
- trackerMode, [522](#)
- trackerPen, [522](#)
- trackerPosition, [523](#)
- trackerRect, [523](#)
- trackerText, [523](#), [524](#)
- trackerWidget, [524](#)
- transform, [524](#)
- transition, [525](#)
- updateDisplay, [525](#)
- widgetKeyPressEvent, [525](#)
- widgetKeyReleaseEvent, [525](#)
- widgetLeaveEvent, [526](#)
- widgetMouseDoubleClickEvent, [526](#)
- widgetMouseMoveEvent, [526](#)
- widgetMousePressEvent, [526](#)
- widgetMouseReleaseEvent, [526](#)
- widgetWheelEvent, [527](#)
- xAxis, [527](#)
- yAxis, [527](#)
- QwtPlotPrintFilter, [527](#)
 - ~QwtPlotPrintFilter, [529](#)
 - apply, [529](#)
 - color, [529](#)
 - font, [530](#)
 - Item, [529](#)
 - Options, [529](#)
 - options, [530](#)
 - QwtPlotPrintFilter, [529](#)
 - reset, [530](#)
 - setOptions, [530](#)
- QwtPlotRasterItem, [531](#)
 - ~QwtPlotRasterItem, [535](#)
 - alpha, [535](#)
 - attach, [535](#)
 - boundingRect, [536](#)
 - CachePolicy, [533](#)
 - cachePolicy, [536](#)
 - detach, [536](#)
 - draw, [536](#)
 - hide, [537](#)
 - invalidateCache, [537](#)
 - invTransform, [537](#)
 - isVisible, [537](#)
 - ItemAttribute, [534](#)
 - itemChanged, [538](#)
 - legendItem, [538](#)
 - paintRect, [538](#)
 - plot, [539](#)
 - QwtPlotRasterItem, [535](#)
 - rasterHint, [539](#)
 - RenderHint, [534](#)
 - renderImage, [539](#)
 - rtti, [539](#)
 - RttiValues, [534](#)
 - scaleRect, [540](#)
 - setAlpha, [540](#)
 - setAxis, [541](#)
 - setCachePolicy, [541](#)
 - setItemAttribute, [541](#)
 - setRenderHint, [541](#)
 - setTitle, [542](#)
 - setVisible, [542](#)
 - setXAxis, [543](#)
 - setYAxis, [543](#)
 - setZ, [543](#)
 - show, [543](#)
 - testItemAttribute, [544](#)
 - testRenderHint, [544](#)
 - title, [544](#)
 - transform, [544](#)
 - updateLegend, [545](#)
 - updateScaleDiv, [545](#)

- xAxis, [546](#)
- yAxis, [546](#)
- z, [546](#)
- QwtPlotRescaler, [546](#)
 - ~QwtPlotRescaler, [549](#)
 - aspectRatio, [549](#)
 - canvas, [549](#)
 - eventFilter, [549](#)
 - expandingDirection, [549](#)
 - expandInterval, [550](#)
 - expandScale, [550](#)
 - interval, [550](#)
 - isEnabled, [550](#)
 - orientation, [551](#)
 - plot, [551](#)
 - QwtPlotRescaler, [548](#)
 - referenceAxis, [551](#)
 - rescale, [551](#), [552](#)
 - RescalePolicy, [548](#)
 - rescalePolicy, [552](#)
 - setAspectRatio, [552](#)
 - setEnabled, [553](#)
 - setExpandingDirection, [553](#)
 - setReferenceAxis, [554](#)
 - setRescalePolicy, [554](#)
 - syncScale, [554](#)
 - updateScales, [554](#)
- QwtPlotScaleItem, [555](#)
 - ~QwtPlotScaleItem, [558](#)
 - attach, [559](#)
 - borderDistance, [559](#)
 - boundingRect, [559](#)
 - detach, [559](#)
 - draw, [560](#)
 - font, [560](#)
 - hide, [560](#)
 - invTransform, [560](#)
 - isScaleDivFromAxis, [561](#)
 - isVisible, [561](#)
 - ItemAttribute, [557](#)
 - itemChanged, [561](#)
 - legendItem, [561](#)
 - paintRect, [561](#)
 - palette, [562](#)
 - plot, [562](#)
 - position, [562](#)
 - QwtPlotScaleItem, [558](#)
 - RenderHint, [558](#)
 - rtti, [562](#)
 - RttiValues, [558](#)
 - scaleDiv, [563](#)
 - scaleDraw, [563](#)
 - scaleRect, [563](#)
 - setAlignment, [564](#)
 - setAxis, [564](#)
 - setBorderDistance, [564](#)
 - setFont, [565](#)
 - setItemAttribute, [565](#)
 - setPalette, [565](#)
 - setPosition, [565](#)
 - setRenderHint, [566](#)
 - setScaleDiv, [566](#)
 - setScaleDivFromAxis, [566](#)
 - setScaleDraw, [567](#)
 - setTitle, [567](#)
 - setVisible, [568](#)
 - setXAxis, [568](#)
 - setYAxis, [568](#)
 - setZ, [568](#)
 - show, [569](#)
 - testItemAttribute, [569](#)
 - testRenderHint, [569](#)
 - title, [570](#)
 - transform, [570](#)
 - updateLegend, [570](#)
 - updateScaleDiv, [571](#)
 - xAxis, [571](#)
 - yAxis, [571](#)
 - z, [571](#)
- QwtPlotSpectrogram, [572](#)
 - ~QwtPlotSpectrogram, [576](#)
 - alpha, [577](#)
 - attach, [577](#)
 - boundingRect, [577](#)
 - CachePolicy, [575](#)
 - cachePolicy, [577](#)
 - colorMap, [578](#)
 - contourLevels, [578](#)
 - contourPen, [578](#)
 - contourRasterSize, [579](#)
 - data, [579](#)
 - defaultContourPen, [579](#)
 - detach, [580](#)
 - DisplayMode, [575](#)
 - draw, [580](#)
 - drawContourLines, [580](#)
 - hide, [581](#)
 - invalidateCache, [581](#)
 - invTransform, [581](#)
 - isVisible, [581](#)

- ItemAttribute, 575
- itemChanged, 582
- legendItem, 582
- paintRect, 582
- plot, 583
- QwtPlotSpectrogram, 576
- rasterHint, 583
- renderContourLines, 583
- RenderHint, 576
- renderImage, 583
- rtti, 584
- RttiValues, 576
- scaleRect, 584
- setAlpha, 584
- setAxis, 585
- setCachePolicy, 585
- setColorMap, 586
- setConrecAttribute, 586
- setContourLevels, 586
- setData, 586
- setDefaultContourPen, 587
- setDisplayMode, 587
- setItemAttribute, 587
- setRenderHint, 588
- setTitle, 588
- setVisible, 588
- setXAxis, 589
- setYAxis, 589
- setZ, 589
- show, 590
- testConrecAttribute, 590
- testDisplayMode, 590
- testItemAttribute, 590
- testRenderHint, 591
- title, 591
- transform, 591
- updateLegend, 592
- updateScaleDiv, 592
- xAxis, 593
- yAxis, 593
- z, 593
- QwtPlotSvgItem, 593
 - ~QwtPlotSvgItem, 596
 - attach, 597
 - boundingRect, 597
 - detach, 597
 - draw, 597
 - hide, 598
 - invTransform, 598
 - isVisible, 598
 - ItemAttribute, 595
 - itemChanged, 598
 - legendItem, 599
 - loadData, 599
 - loadFile, 599
 - paintRect, 600
 - plot, 600
 - QwtPlotSvgItem, 596
 - render, 600
 - RenderHint, 595
 - rtti, 600
 - RttiValues, 596
 - scaleRect, 600
 - setAxis, 601
 - setItemAttribute, 601
 - setRenderHint, 601
 - setTitle, 602
 - setVisible, 602
 - setXAxis, 602
 - setYAxis, 603
 - setZ, 603
 - show, 603
 - testItemAttribute, 603
 - testRenderHint, 604
 - title, 604
 - transform, 604
 - updateLegend, 605
 - updateScaleDiv, 605
 - viewBox, 606
 - xAxis, 606
 - yAxis, 606
 - z, 606
- QwtPlotZoomer, 606
 - accept, 618
 - append, 618
 - appended, 618, 619
 - begin, 619
 - canvas, 619
 - changed, 619
 - DisplayMode, 611
 - drawRubberBand, 620
 - drawTracker, 620
 - end, 620
 - eventFilter, 620
 - initKeyPattern, 621
 - initMousePattern, 621
 - invTransform, 621
 - isActive, 622
 - isEnabled, 622
 - keyMatch, 622, 623

- keyPattern, [623](#)
- KeyPatternCode, [612](#)
- maxStackDepth, [623](#)
- minZoomSize, [624](#)
- mouseMatch, [624](#)
- mousePattern, [625](#)
- MousePatternCode, [612](#)
- move, [625](#)
- moveBy, [626](#)
- moved, [626](#)
- parentWidget, [627](#)
- pickRect, [627](#)
- plot, [627](#)
- QwtPlotZoomer, [616](#), [617](#)
- RectSelectionType, [614](#)
- rescale, [627](#)
- reset, [627](#)
- SizeMode, [614](#)
- resizeMode, [628](#)
- RubberBand, [615](#)
- rubberBand, [628](#)
- rubberBandPen, [628](#)
- rubberBandWidget, [628](#)
- scaleRect, [628](#)
- selected, [629](#)
- selection, [629](#)
- selectionFlags, [630](#)
- SelectionMode, [615](#)
- SelectionType, [616](#)
- setAxis, [630](#)
- setEnabled, [630](#)
- setKeyPattern, [630](#), [631](#)
- setMaxStackDepth, [631](#)
- setMousePattern, [631](#)
- setSizeMode, [632](#)
- setRubberBand, [632](#)
- setRubberBandPen, [632](#)
- setSelectionFlags, [633](#)
- setTrackerFont, [633](#)
- setTrackerMode, [633](#)
- setTrackerPen, [634](#)
- setZoomBase, [634](#), [635](#)
- setZoomStack, [635](#)
- stateMachine, [635](#)
- stretchSelection, [636](#)
- trackerFont, [636](#)
- trackerMode, [636](#)
- trackerPen, [637](#)
- trackerPosition, [637](#)
- trackerRect, [637](#)
- trackerText, [637](#), [638](#)
- trackerWidget, [638](#)
- transform, [638](#)
- transition, [639](#)
- updateDisplay, [639](#)
- widgetKeyPressEvent, [639](#)
- widgetKeyReleaseEvent, [639](#)
- widgetLeaveEvent, [640](#)
- widgetMouseDoubleClickEvent, [640](#)
- widgetMouseMoveEvent, [640](#)
- widgetMousePressEvent, [640](#)
- widgetMouseReleaseEvent, [641](#)
- widgetWheelEvent, [641](#)
- xAxis, [641](#)
- yAxis, [641](#)
- zoom, [642](#)
- zoomBase, [642](#)
- zoomed, [643](#)
- zoomRect, [643](#)
- zoomRectIndex, [643](#)
- zoomStack, [643](#)
- QwtPolygonFData, [643](#)
 - boundingRect, [645](#)
 - copy, [645](#)
 - data, [645](#)
 - operator=, [645](#)
 - QwtPolygonFData, [644](#)
 - size, [645](#)
 - x, [645](#)
 - y, [646](#)
- QwtRasterData, [646](#)
 - ~QwtRasterData, [648](#)
 - boundingRect, [648](#)
 - ConrecAttribute, [647](#)
 - contourLines, [648](#)
 - copy, [648](#)
 - discardRaster, [649](#)
 - initRaster, [649](#)
 - QwtRasterData, [647](#), [648](#)
 - range, [649](#)
 - rasterHint, [649](#)
 - setBoundingRect, [650](#)
 - value, [650](#)
- QwtRichTextEngine, [651](#)
 - draw, [652](#)
 - heightForWidth, [652](#)
 - mightRender, [652](#)
 - QwtRichTextEngine, [651](#)
 - textMargins, [652](#)
 - textSize, [653](#)

- QwtRoundScaleDraw, 653
 - ~QwtRoundScaleDraw, 656
 - center, 656
 - draw, 656
 - drawBackbone, 656
 - drawLabel, 657
 - drawTick, 657
 - enableComponent, 657
 - extent, 658
 - hasComponent, 658
 - invalidateCache, 658
 - label, 658
 - majTickLength, 659
 - map, 659
 - minimumExtent, 659
 - moveCenter, 659, 660
 - operator=, 660
 - QwtRoundScaleDraw, 656
 - radius, 660
 - ScaleComponent, 655
 - scaleDiv, 660
 - scaleMap, 660
 - setAngleRange, 660
 - setMinimumExtent, 661
 - setRadius, 661
 - setScaleDiv, 661
 - setSpacing, 662
 - setTickLength, 662
 - setTransformation, 662
 - spacing, 663
 - tickLabel, 663
 - tickLength, 663
- QwtScaleArithmetic, 663
 - ceil125, 664
 - ceilEps, 664
 - compareEps, 664
 - divideEps, 665
 - floor125, 665
 - floorEps, 665
- QwtScaleDiv, 666
 - contains, 667
 - interval, 668
 - invalidate, 668
 - invert, 668
 - isValid, 668
 - lowerBound, 668
 - operator==, 669
 - QwtScaleDiv, 667
 - range, 669
 - setInterval, 669
 - setTicks, 669
 - ticks, 670
 - TickType, 667
 - upperBound, 670
- QwtScaleDraw, 670
 - ~QwtScaleDraw, 673
 - Alignment, 672
 - alignment, 673
 - boundingLabelRect, 673
 - draw, 674
 - drawBackbone, 674
 - drawLabel, 674
 - drawTick, 675
 - enableComponent, 675
 - extent, 675
 - getBorderDistHint, 676
 - hasComponent, 676
 - invalidateCache, 676
 - label, 676
 - labelAlignment, 677
 - labelMatrix, 677
 - labelPosition, 677
 - labelRect, 678
 - labelRotation, 678
 - labelSize, 678
 - length, 678
 - majTickLength, 678
 - map, 679
 - maxLabelHeight, 679
 - maxLabelWidth, 679
 - minimumExtent, 679
 - minLabelDist, 679
 - minLength, 680
 - move, 680
 - operator=, 681
 - orientation, 681
 - pos, 681
 - QwtScaleDraw, 673
 - ScaleComponent, 672
 - scaleDiv, 681
 - scaleMap, 682
 - setAlignment, 682
 - setLabelAlignment, 682
 - setLabelRotation, 683
 - setLength, 683
 - setMinimumExtent, 683
 - setScaleDiv, 684
 - setSpacing, 684
 - setTickLength, 684
 - setTransformation, 684

- spacing, 685
- tickLabel, 685
- tickLength, 685
- QwtScaleEngine, 685
 - ~QwtScaleEngine, 687
 - Attribute, 687
 - attributes, 688
 - autoScale, 688
 - buildInterval, 688
 - contains, 688
 - divideInterval, 689
 - divideScale, 689
 - lowerMargin, 689
 - QwtScaleEngine, 687
 - reference, 689
 - setAttribute, 690
 - setAttributes, 690
 - setMargins, 690
 - setReference, 691
 - strip, 691
 - testAttribute, 691
 - transformation, 692
 - upperMargin, 692
- QwtScaleMap, 692
 - ~QwtScaleMap, 693
 - invTransform, 693
 - operator=, 694
 - p1, 694
 - p2, 694
 - pDist, 694
 - QwtScaleMap, 693
 - s1, 694
 - s2, 694
 - sDist, 694
 - setPaintInterval, 695
 - setPaintXInterval, 695
 - setScaleInterval, 695
 - setTransformation, 695
 - transform, 696
 - transformation, 696
 - xTransform, 696
- QwtScaleTransformation, 696
 - ~QwtScaleTransformation, 697
 - copy, 697
 - invXForm, 697
 - QwtScaleTransformation, 697
 - type, 698
 - xForm, 698
- QwtScaleWidget, 698
 - ~QwtScaleWidget, 700
 - alignment, 701
 - dimForLength, 701
 - draw, 701
 - drawTitle, 701
 - endBorderDist, 701
 - getBorderDistHint, 702
 - getMinBorderDist, 702
 - layoutScale, 702
 - margin, 702
 - minimumSizeHint, 703
 - paintEvent, 703
 - penWidth, 703
 - QwtScaleWidget, 700
 - resizeEvent, 703
 - scaleChange, 703
 - scaleDivChanged, 704
 - scaleDraw, 704
 - setAlignment, 704
 - setBorderDist, 704
 - setLabelAlignment, 705
 - setLabelRotation, 705
 - setMargin, 705
 - setMinBorderDist, 705
 - setPenWidth, 706
 - setScaleDiv, 706
 - setScaleDraw, 706
 - setSpacing, 707
 - setTitle, 707
 - sizeHint, 708
 - spacing, 708
 - startBorderDist, 708
 - title, 708
 - titleHeightForWidth, 708
- QwtSimpleCompassRose, 709
 - draw, 710
 - drawRose, 710
 - numThornLevels, 711
 - numThorns, 711
 - palette, 711
 - QwtSimpleCompassRose, 710
 - setNumThornLevels, 712
 - setNumThorns, 712
 - setPalette, 712
 - setWidth, 712
 - width, 712
- QwtSlider, 713
 - abstractScaleDraw, 717, 718
 - autoScale, 718
 - BGSTYLE, 716
 - bgStyle, 718

- borderWidth, 718
- draw, 718
- drawSlider, 718
- drawThumb, 719
- exactPrevValue, 719
- exactValue, 719
- fitValue, 719
- fontChange, 720
- getScrollMode, 720
- getValue, 720
- incPages, 720
- incValue, 721
- isReadOnly, 721
- isValid, 721
- keyPressEvent, 721
- layoutSlider, 722
- mass, 722
- maxValue, 722
- minimumSizeHint, 723
- minValue, 723
- mouseMoveEvent, 723
- mousePressEvent, 723
- mouseReleaseEvent, 724
- orientation, 724
- pageSize, 724
- paintEvent, 724
- periodic, 724
- prevValue, 724
- QwtSlider, 717
- rangeChange, 725
- rescale, 725
- resizeEvent, 725
- scaleChange, 725
- scaleDraw, 725, 726
- scaleEngine, 726
- scaleMap, 726
- scaleMaxMajor, 727
- scaleMaxMinor, 727
- ScalePos, 716
- scalePosition, 727
- ScrollMode, 717
- setAbstractScaleDraw, 727
- setAutoScale, 727
- setBgStyle, 728
- setBorderWidth, 728
- setMargins, 728
- setMass, 728
- setOrientation, 729
- setPeriodic, 729
- setPosition, 729
- setRange, 729
- setReadOnly, 730
- setScale, 730, 731
- setScaleDraw, 731
- setScaleEngine, 731
- setScaleMaxMajor, 732
- setScaleMaxMinor, 732
- setScalePosition, 732
- setStep, 733
- setThumbLength, 733
- setThumbWidth, 733
- setTracking, 734
- setUpdateTime, 734
- setValid, 734
- setValue, 735
- sizeHint, 735
- sliderMoved, 735
- sliderPressed, 735
- sliderReleased, 735
- step, 736
- stepChange, 736
- stopMoving, 736
- thumbLength, 736
- thumbWidth, 736
- timerEvent, 736
- value, 737
- valueChange, 737
- valueChanged, 737
- wheelEvent, 737
- xyPosition, 737
- QwtSpline, 738
 - ~QwtSpline, 740
 - buildNaturalSpline, 740
 - buildPeriodicSpline, 740
 - coefficientsA, 740
 - coefficientsB, 740
 - coefficientsC, 741
 - isValid, 741
 - operator=, 741
 - points, 741
 - QwtSpline, 739
 - reset, 741
 - setPoints, 741
 - setSplineType, 742
 - SplineType, 739
 - splineType, 742
 - value, 742
- QwtSplineCurveFitter, 742
 - ~QwtSplineCurveFitter, 744
 - fitCurve, 744

- fitMode, [744](#)
- QwtSplineCurveFitter, [744](#)
- setFitMode, [744](#)
- setSplineSize, [745](#)
- splineSize, [745](#)
- QwtSymbol, [745](#)
 - ~QwtSymbol, [747](#)
 - brush, [747](#)
 - clone, [747](#)
 - draw, [747](#), [748](#)
 - operator==, [748](#)
 - pen, [748](#)
 - QwtSymbol, [747](#)
 - setBrush, [748](#)
 - setPen, [749](#)
 - setSize, [749](#)
 - setStyle, [749](#)
 - size, [750](#)
 - Style, [746](#)
 - style, [750](#)
- QwtText, [751](#)
 - ~QwtText, [754](#)
 - backgroundBrush, [755](#)
 - backgroundPen, [755](#)
 - color, [755](#)
 - draw, [755](#)
 - font, [755](#)
 - heightForWidth, [755](#)
 - isEmpty, [756](#)
 - isNull, [756](#)
 - LayoutAttribute, [753](#)
 - operator=, [756](#)
 - operator==, [756](#)
 - PaintAttribute, [753](#)
 - QwtText, [754](#)
 - renderFlags, [756](#)
 - setBackgroundBrush, [757](#)
 - setBackgroundPen, [757](#)
 - setColor, [757](#)
 - setFont, [757](#)
 - setLayoutAttribute, [758](#)
 - setPaintAttribute, [758](#)
 - setRenderFlags, [758](#)
 - setText, [759](#)
 - setTextEngine, [759](#)
 - testLayoutAttribute, [760](#)
 - testPaintAttribute, [760](#)
 - text, [760](#)
 - textEngine, [760](#), [761](#)
 - TextFormat, [753](#)
 - textSize, [761](#)
 - usedColor, [762](#)
 - usedFont, [762](#)
- QwtTextEngine, [762](#)
 - ~QwtTextEngine, [763](#)
 - draw, [764](#)
 - heightForWidth, [764](#)
 - mightRender, [764](#)
 - QwtTextEngine, [763](#)
 - textMargins, [764](#)
 - textSize, [765](#)
- QwtTextLabel, [765](#)
 - ~QwtTextLabel, [767](#)
 - clear, [767](#)
 - drawContents, [767](#)
 - drawText, [767](#)
 - heightForWidth, [768](#)
 - indent, [768](#)
 - margin, [768](#)
 - minimumSizeHint, [768](#)
 - paintEvent, [768](#)
 - QwtTextLabel, [767](#)
 - setIndent, [768](#)
 - setMargin, [769](#)
 - setText, [769](#)
 - sizeHint, [769](#)
 - text, [769](#)
 - textRect, [770](#)
- QwtThermo, [770](#)
 - ~QwtThermo, [773](#)
 - abstractScaleDraw, [773](#)
 - alarmBrush, [774](#)
 - alarmColor, [774](#)
 - alarmEnabled, [774](#)
 - alarmLevel, [774](#)
 - autoScale, [774](#)
 - borderWidth, [774](#)
 - draw, [775](#)
 - drawThermo, [775](#)
 - fillBrush, [775](#)
 - fillColor, [775](#)
 - fontChange, [775](#)
 - layoutThermo, [776](#)
 - maxValue, [776](#)
 - minimumSizeHint, [776](#)
 - minValue, [776](#)
 - paintEvent, [776](#)
 - pipeWidth, [776](#)
 - QwtThermo, [773](#)
 - rescale, [777](#)

- resizeEvent, [777](#)
- scaleChange, [777](#)
- scaleDraw, [777](#)
- scaleEngine, [778](#)
- scaleMap, [778](#)
- scaleMaxMajor, [778](#)
- scaleMaxMinor, [778](#)
- scalePosition, [779](#)
- setAbstractScaleDraw, [779](#)
- setAlarmBrush, [779](#)
- setAlarmColor, [779](#)
- setAlarmEnabled, [780](#)
- setAlarmLevel, [780](#)
- setAutoScale, [780](#)
- setBorderWidth, [780](#)
- setFillBrush, [780](#)
- setFillColor, [781](#)
- setMargin, [781](#)
- setMaxValue, [781](#)
- setMinValue, [782](#)
- setOrientation, [782](#)
- setPipeWidth, [782](#)
- setRange, [783](#)
- setScale, [783](#), [784](#)
- setScaleDraw, [784](#)
- setScaleEngine, [784](#)
- setScaleMaxMajor, [784](#)
- setScaleMaxMinor, [785](#)
- setScalePosition, [785](#)
- setValue, [786](#)
- sizeHint, [786](#)
- value, [786](#)
- QwtWheel, [786](#)
 - ~QwtWheel, [789](#)
 - draw, [790](#)
 - drawWheel, [790](#)
 - drawWheelBackground, [790](#)
 - exactPrevValue, [790](#)
 - exactValue, [790](#)
 - fitValue, [791](#)
 - getScrollMode, [791](#)
 - getValue, [791](#)
 - incPages, [792](#)
 - incValue, [792](#)
 - internalBorder, [792](#)
 - isReadOnly, [792](#)
 - isValid, [793](#)
 - keyPressEvent, [793](#)
 - layoutWheel, [793](#)
 - mass, [793](#)
 - maxValue, [794](#)
 - minimumSizeHint, [794](#)
 - minValue, [794](#)
 - mouseMoveEvent, [794](#)
 - mousePressEvent, [795](#)
 - mouseReleaseEvent, [795](#)
 - orientation, [795](#)
 - pageSize, [795](#)
 - paintEvent, [795](#)
 - paletteChange, [795](#)
 - periodic, [796](#)
 - prevValue, [796](#)
 - QwtWheel, [789](#)
 - rangeChange, [796](#)
 - resizeEvent, [796](#)
 - ScrollMode, [789](#)
 - setColorArray, [796](#)
 - setInternalBorder, [796](#)
 - setMass, [797](#)
 - setOrientation, [797](#)
 - setPeriodic, [797](#)
 - setPosition, [798](#)
 - setRange, [798](#)
 - setReadOnly, [798](#)
 - setStep, [799](#)
 - setTickCnt, [799](#)
 - setTotalAngle, [799](#)
 - setTracking, [800](#)
 - setUpdateTime, [800](#)
 - setValid, [800](#)
 - setValue, [801](#)
 - setViewAngle, [801](#)
 - setWheelWidth, [801](#)
 - sizeHint, [802](#)
 - sliderMoved, [802](#)
 - sliderPressed, [802](#)
 - sliderReleased, [802](#)
 - step, [802](#)
 - stepChange, [802](#)
 - stopMoving, [803](#)
 - tickCnt, [803](#)
 - timerEvent, [803](#)
 - totalAngle, [803](#)
 - value, [803](#)
 - valueChange, [804](#)
 - valueChanged, [804](#)
 - viewAngle, [804](#)
 - wheelEvent, [804](#)
- radius

- QwtDialScaleDraw, [189](#)
- QwtRoundScaleDraw, [660](#)
- range
 - QwtRasterData, [649](#)
 - QwtScaleDiv, [669](#)
- rangeChange
 - QwtAbstractSlider, [46](#)
 - QwtAnalogClock, [73](#)
 - QwtCompass, [112](#)
 - QwtCounter, [142](#)
 - QwtDial, [169](#)
 - QwtDoubleRange, [209](#)
 - QwtSlider, [725](#)
 - QwtWheel, [796](#)
- rasterHint
 - QwtPlotRasterItem, [539](#)
 - QwtPlotSpectrogram, [583](#)
 - QwtRasterData, [649](#)
- RectSelectionType
 - QwtPicker, [323](#)
 - QwtPlotPicker, [502](#)
 - QwtPlotZoomer, [614](#)
- reference
 - QwtLinearScaleEngine, [282](#)
 - QwtLog10ScaleEngine, [289](#)
 - QwtScaleEngine, [689](#)
- referenceAxis
 - QwtPlotRescaler, [551](#)
- released
 - QwtLegendItem, [265](#)
- remove
 - QwtLegend, [257](#)
- render
 - QwtPlotSvgItem, [600](#)
- renderContourLines
 - QwtPlotSpectrogram, [583](#)
- renderFlags
 - QwtText, [756](#)
- RenderHint
 - QwtPlotCurve, [397](#)
 - QwtPlotGrid, [424](#)
 - QwtPlotItem, [440](#)
 - QwtPlotMarker, [473](#)
 - QwtPlotRasterItem, [534](#)
 - QwtPlotScaleItem, [558](#)
 - QwtPlotSpectrogram, [576](#)
 - QwtPlotSvgItem, [595](#)
- renderImage
 - QwtPlotRasterItem, [539](#)
 - QwtPlotSpectrogram, [583](#)
- replot
 - QwtPlot, [377](#)
 - QwtPlotCanvas, [389](#)
- rescale
 - QwtAbstractScale, [25](#)
 - QwtKnob, [240](#)
 - QwtMagnifier, [295](#)
 - QwtPlotMagnifier, [464](#)
 - QwtPlotRescaler, [551](#), [552](#)
 - QwtPlotZoomer, [627](#)
 - QwtSlider, [725](#)
 - QwtThermo, [777](#)
- RescalePolicy
 - QwtPlotRescaler, [548](#)
- rescalePolicy
 - QwtPlotRescaler, [552](#)
- reset
 - QwtPicker, [333](#)
 - QwtPickerClickPointMachine, [345](#)
 - QwtPickerClickRectMachine, [347](#)
 - QwtPickerDragPointMachine, [349](#)
 - QwtPickerDragRectMachine, [351](#)
 - QwtPickerMachine, [353](#)
 - QwtPickerPolygonMachine, [355](#)
 - QwtPlotPicker, [515](#)
 - QwtPlotPrintFilter, [530](#)
 - QwtPlotZoomer, [627](#)
 - QwtSpline, [741](#)
- resetMetricsMap
 - QwtPainter, [309](#)
- resizeEvent
 - QwtAnalogClock, [73](#)
 - QwtCompass, [113](#)
 - QwtDial, [169](#)
 - QwtKnob, [240](#)
 - QwtLegend, [257](#)
 - QwtPlot, [378](#)
 - QwtScaleWidget, [703](#)
 - QwtSlider, [725](#)
 - QwtThermo, [777](#)
 - QwtWheel, [796](#)
- SizeMode
 - QwtPicker, [324](#)
 - QwtPlotPicker, [503](#)
 - QwtPlotZoomer, [614](#)
- resizeMode
 - QwtPicker, [334](#)
 - QwtPlotPicker, [515](#)
 - QwtPlotZoomer, [628](#)
- rgb

- QwtAlphaColorMap, 55
- QwtColorMap, 96
- QwtLinearColorMap, 277
- rose
 - QwtCompass, 113
- rtti
 - QwtPlotCurve, 407
 - QwtPlotGrid, 429
 - QwtPlotItem, 444
 - QwtPlotMarker, 478
 - QwtPlotRasterItem, 539
 - QwtPlotScaleItem, 562
 - QwtPlotSpectrogram, 584
 - QwtPlotSvgItem, 600
- RttiValues
 - QwtPlotCurve, 397
 - QwtPlotGrid, 424
 - QwtPlotItem, 440
 - QwtPlotMarker, 473
 - QwtPlotRasterItem, 534
 - QwtPlotScaleItem, 558
 - QwtPlotSpectrogram, 576
 - QwtPlotSvgItem, 596
- RubberBand
 - QwtPicker, 324
 - QwtPlotPicker, 503
 - QwtPlotZoomer, 615
- rubberBand
 - QwtPicker, 334
 - QwtPlotPicker, 515
 - QwtPlotZoomer, 628
- rubberBandPen
 - QwtPicker, 334
 - QwtPlotPicker, 515
 - QwtPlotZoomer, 628
- rubberBandWidget
 - QwtPicker, 334
 - QwtPlotPicker, 516
 - QwtPlotZoomer, 628
- s1
 - QwtScaleMap, 694
- s2
 - QwtScaleMap, 694
- scaleChange
 - QwtAbstractScale, 26
 - QwtScaleWidget, 703
 - QwtSlider, 725
 - QwtThermo, 777
- ScaleComponent
 - QwtAbstractScaleDraw, 31
 - QwtDialScaleDraw, 185
 - QwtRoundScaleDraw, 655
 - QwtScaleDraw, 672
- scaleContentsRect
 - QwtAnalogClock, 73
 - QwtCompass, 113
 - QwtDial, 169
- scaleDiv
 - QwtAbstractScaleDraw, 35
 - QwtDialScaleDraw, 189
 - QwtPlotScaleItem, 563
 - QwtRoundScaleDraw, 660
 - QwtScaleDraw, 681
- scaleDivChanged
 - QwtScaleWidget, 704
- scaledPen
 - QwtPainter, 309
- scaleDraw
 - QwtAnalogClock, 73
 - QwtCompass, 113, 114
 - QwtDial, 170
 - QwtKnob, 240
 - QwtPlotScaleItem, 563
 - QwtScaleWidget, 704
 - QwtSlider, 725, 726
 - QwtThermo, 777
- scaleEngine
 - QwtAbstractScale, 26
 - QwtKnob, 241
 - QwtSlider, 726
 - QwtThermo, 778
- scaleLabel
 - QwtAnalogClock, 73
 - QwtCompass, 114
 - QwtDial, 170
- scaleMap
 - QwtAbstractScale, 26
 - QwtAbstractScaleDraw, 35
 - QwtDialScaleDraw, 189
 - QwtKnob, 241
 - QwtRoundScaleDraw, 660
 - QwtScaleDraw, 682
 - QwtSlider, 726
 - QwtThermo, 778
- scaleMaxMajor
 - QwtAbstractScale, 26
 - QwtKnob, 241
 - QwtSlider, 727
 - QwtThermo, 778

- scaleMaxMinor
 - QwtAbstractScale, [27](#)
 - QwtKnob, [242](#)
 - QwtSlider, [727](#)
 - QwtThermo, [778](#)
- ScaleOptions
 - QwtAnalogClock, [60](#)
 - QwtCompass, [100](#)
 - QwtDial, [158](#)
- ScalePos
 - QwtSlider, [716](#)
- scalePosition
 - QwtSlider, [727](#)
 - QwtThermo, [779](#)
- scaleRect
 - QwtPlotCurve, [407](#)
 - QwtPlotGrid, [429](#)
 - QwtPlotItem, [444](#)
 - QwtPlotLayout, [456](#)
 - QwtPlotMarker, [478](#)
 - QwtPlotPicker, [516](#)
 - QwtPlotRasterItem, [540](#)
 - QwtPlotScaleItem, [563](#)
 - QwtPlotSpectrogram, [584](#)
 - QwtPlotSvgItem, [600](#)
 - QwtPlotZoomer, [628](#)
- ScrollMode
 - QwtAbstractSlider, [40](#)
 - QwtAnalogClock, [60](#)
 - QwtCompass, [101](#)
 - QwtDial, [158](#)
 - QwtKnob, [233](#)
 - QwtSlider, [717](#)
 - QwtWheel, [789](#)
- sDist
 - QwtScaleMap, [694](#)
- selected
 - QwtPicker, [334](#)
 - QwtPlotPicker, [516](#), [517](#)
 - QwtPlotZoomer, [629](#)
- selection
 - QwtPicker, [335](#)
 - QwtPlotPicker, [517](#)
 - QwtPlotZoomer, [629](#)
- selectionFlags
 - QwtPicker, [335](#)
 - QwtPlotPicker, [517](#)
 - QwtPlotZoomer, [630](#)
- SelectionMode
 - QwtPicker, [325](#)
- QwtPlotPicker, [504](#)
- QwtPlotZoomer, [615](#)
- SelectionType
 - QwtPicker, [325](#)
 - QwtPlotPicker, [504](#)
 - QwtPlotZoomer, [616](#)
- setAbortKey
 - QwtPanner, [314](#)
 - QwtPlotPanner, [493](#)
- setAbstractScaleDraw
 - QwtAbstractScale, [27](#)
 - QwtKnob, [242](#)
 - QwtSlider, [727](#)
 - QwtThermo, [779](#)
- setAlarmBrush
 - QwtThermo, [779](#)
- setAlarmColor
 - QwtThermo, [779](#)
- setAlarmEnabled
 - QwtThermo, [780](#)
- setAlarmLevel
 - QwtThermo, [780](#)
- setAlignCanvasToScales
 - QwtPlotLayout, [456](#)
- setAlignment
 - QwtPlotScaleItem, [564](#)
 - QwtScaleDraw, [682](#)
 - QwtScaleWidget, [704](#)
- setAlpha
 - QwtPlotRasterItem, [540](#)
 - QwtPlotSpectrogram, [584](#)
- setAngleRange
 - QwtDialScaleDraw, [190](#)
 - QwtRoundScaleDraw, [660](#)
- setAspectRatio
 - QwtPlotRescaler, [552](#)
- setAttribute
 - QwtLinearScaleEngine, [282](#)
 - QwtLog10ScaleEngine, [289](#)
 - QwtScaleEngine, [690](#)
- setAttributes
 - QwtLinearScaleEngine, [282](#)
 - QwtLog10ScaleEngine, [289](#)
 - QwtScaleEngine, [690](#)
- setAutoDelete
 - QwtPlot, [378](#)
 - QwtPlotDict, [421](#)
- setAutoReplot
 - QwtPlot, [378](#)
- setAutoScale

- QwtAbstractScale, 27
- QwtKnob, 242
- QwtSlider, 727
- QwtThermo, 780
- setAxis
 - QwtPlotCurve, 408
 - QwtPlotGrid, 429
 - QwtPlotItem, 445
 - QwtPlotMarker, 478
 - QwtPlotPicker, 517
 - QwtPlotRasterItem, 541
 - QwtPlotScaleItem, 564
 - QwtPlotSpectrogram, 585
 - QwtPlotSvgItem, 601
 - QwtPlotZoomer, 630
- setAxisAutoScale
 - QwtPlot, 378
- setAxisEnabled
 - QwtPlotMagnifier, 464
 - QwtPlotPanner, 493
- setAxisFont
 - QwtPlot, 379
- setAxisLabelAlignment
 - QwtPlot, 379
- setAxisLabelRotation
 - QwtPlot, 379
- setAxisMaxMajor
 - QwtPlot, 380
- setAxisMaxMinor
 - QwtPlot, 380
- setAxisScale
 - QwtPlot, 380
- setAxisScaleDiv
 - QwtPlot, 381
- setAxisScaleDraw
 - QwtPlot, 381
- setAxisScaleEngine
 - QwtPlot, 381
- setAxisTitle
 - QwtPlot, 382
- setBackgroundBrush
 - QwtText, 757
- setBackgroundPen
 - QwtText, 757
- setBaseline
 - QwtPlotCurve, 408
- setBgStyle
 - QwtSlider, 728
- setBorderDist
 - QwtScaleWidget, 704
- setBorderDistance
 - QwtPlotScaleItem, 564
- setBorderFlags
 - QwtDoubleInterval, 203
- setBorderWidth
 - QwtKnob, 242
 - QwtSlider, 728
 - QwtThermo, 780
- setBoundingRect
 - QwtRasterData, 650
- setBrush
 - QwtPlotCurve, 408
 - QwtSymbol, 748
- setCachePolicy
 - QwtPlotRasterItem, 541
 - QwtPlotSpectrogram, 585
- setCanvasBackground
 - QwtPlot, 382
- setCanvasLineWidth
 - QwtPlot, 382
- setCanvasMargin
 - QwtPlotLayout, 457
- setChecked
 - QwtLegendItem, 265
- setClipRect
 - QwtPainter, 309
- setColor
 - QwtAlphaColorMap, 55
 - QwtText, 757
- setColorArray
 - QwtWheel, 796
- setColorInterval
 - QwtLinearColorMap, 277
- setColorMap
 - QwtPlotSpectrogram, 586
- setConrecAttribute
 - QwtPlotSpectrogram, 586
- setContourLevels
 - QwtPlotSpectrogram, 586
- setCurrentTime
 - QwtAnalogClock, 74
- setCursor
 - QwtPanner, 314
 - QwtPlotPanner, 493
- setCurveAttribute
 - QwtPlotCurve, 409
- setCurveFitter
 - QwtPlotCurve, 409
- setCurvePen
 - QwtLegendItem, 266

- setCurveType
 - QwtPlotCurve, 409
- setData
 - QwtIntervalData, 229
 - QwtPlotCurve, 410
 - QwtPlotSpectrogram, 586
- setDefaultContourPen
 - QwtPlotSpectrogram, 587
- setDeviceClipping
 - QwtPainter, 309
- setDirection
 - QwtAnalogClock, 74
 - QwtCompass, 114
 - QwtDial, 170
- setDisplayMode
 - QwtPlotSpectrogram, 587
- setDisplayPolicy
 - QwtLegend, 258
- setDown
 - QwtLegendItem, 266
- setEditable
 - QwtCounter, 142
- setEnabled
 - QwtMagnifier, 295
 - QwtPanner, 314
 - QwtPicker, 335
 - QwtPlotMagnifier, 464
 - QwtPlotPanner, 494
 - QwtPlotPicker, 517
 - QwtPlotRescaler, 553
 - QwtPlotZoomer, 630
- setExpandingDirection
 - QwtPlotRescaler, 553
- setExpandingDirections
 - QwtDynGridLayout, 217
- setFillBrush
 - QwtThermo, 780
- setFillColor
 - QwtThermo, 781
- setFitMode
 - QwtSplineCurveFitter, 744
- setFocusIndicator
 - QwtPlotCanvas, 389
- setFont
 - QwtPlotScaleItem, 565
 - QwtText, 757
- setFrameShadow
 - QwtAnalogClock, 74
 - QwtCompass, 114
 - QwtDial, 170
- setGeometry
 - QwtDynGridLayout, 217
- setHand
 - QwtAnalogClock, 74
- setIdentifierMode
 - QwtLegendItem, 266
- setIdentifierWidth
 - QwtLegendItem, 266
- setIncSteps
 - QwtCounter, 142
- setIndent
 - QwtLegendItem, 266
 - QwtTextLabel, 768
- setInternalBorder
 - QwtWheel, 796
- setInterval
 - QwtDoubleInterval, 203
 - QwtScaleDiv, 669
- setItemAttribute
 - QwtPlotCurve, 411
 - QwtPlotGrid, 430
 - QwtPlotItem, 445
 - QwtPlotMarker, 479
 - QwtPlotRasterItem, 541
 - QwtPlotScaleItem, 565
 - QwtPlotSpectrogram, 587
 - QwtPlotSvgItem, 601
- setItemMode
 - QwtLegend, 258
 - QwtLegendItem, 267
- setKeyFactor
 - QwtMagnifier, 296
 - QwtPlotMagnifier, 465
- setKeyPattern
 - QwtEventPattern, 226
 - QwtPicker, 335
 - QwtPlotPicker, 518
 - QwtPlotZoomer, 630, 631
- setKnobWidth
 - QwtKnob, 242
- setLabel
 - QwtPlotMarker, 479
- setLabelAlignment
 - QwtPlotMarker, 479
 - QwtScaleDraw, 682
 - QwtScaleWidget, 705
- setLabelMap
 - QwtCompass, 115
- setLabelOrientation
 - QwtPlotMarker, 480

- setLabelRotation
 - QwtScaleDraw, 683
 - QwtScaleWidget, 705
- setLayoutAttribute
 - QwtText, 758
- setLegendPosition
 - QwtPlotLayout, 457
- setLegendRatio
 - QwtPlotLayout, 458
- setLength
 - QwtScaleDraw, 683
- setLinePen
 - QwtPlotMarker, 480
- setLineStyle
 - QwtPlotMarker, 480
- setLineWidth
 - QwtAnalogClock, 75
 - QwtCompass, 115
 - QwtDial, 171
- setMajPen
 - QwtPlotGrid, 430
- setMargin
 - QwtLegendItem, 267
 - QwtPlot, 383
 - QwtPlotLayout, 458
 - QwtScaleWidget, 705
 - QwtTextLabel, 769
 - QwtThermo, 781
- setMargins
 - QwtLinearScaleEngine, 282
 - QwtLog10ScaleEngine, 290
 - QwtScaleEngine, 690
 - QwtSlider, 728
- setMass
 - QwtAbstractSlider, 46
 - QwtAnalogClock, 75
 - QwtCompass, 115
 - QwtDial, 171
 - QwtKnob, 243
 - QwtSlider, 728
 - QwtWheel, 797
- setMaxCols
 - QwtDynGridLayout, 218
- setMaxStackDepth
 - QwtPlotZoomer, 631
- setMaxValue
 - QwtCounter, 142
 - QwtDoubleInterval, 204
 - QwtThermo, 781
- setMetricsMap
 - QwtPainter, 309, 310
- setMinBorderDist
 - QwtScaleWidget, 705
- setMinimumExtent
 - QwtAbstractScaleDraw, 35
 - QwtDialScaleDraw, 190
 - QwtRoundScaleDraw, 661
 - QwtScaleDraw, 683
- setMinPen
 - QwtPlotGrid, 430
- setMinValue
 - QwtCounter, 143
 - QwtDoubleInterval, 204
 - QwtThermo, 782
- setMode
 - QwtAnalogClock, 75
 - QwtCompass, 116
 - QwtDial, 171
 - QwtLinearColorMap, 277
- setMouseButton
 - QwtMagnifier, 296
 - QwtPanner, 314
 - QwtPlotMagnifier, 465
 - QwtPlotPanner, 494
- setMouseFactor
 - QwtMagnifier, 296
 - QwtPlotMagnifier, 465
- setMousePattern
 - QwtEventPattern, 227
 - QwtPicker, 336
 - QwtPlotPicker, 518
 - QwtPlotZoomer, 631
- setNeedle
 - QwtCompass, 116
 - QwtDial, 172
- setNumButtons
 - QwtCounter, 143
- setNumThornLevels
 - QwtSimpleCompassRose, 712
- setNumThorns
 - QwtSimpleCompassRose, 712
- setOptions
 - QwtPlotPrintFilter, 530
- setOrientation
 - QwtAbstractSlider, 46
 - QwtAnalogClock, 76
 - QwtCompass, 116
 - QwtDial, 172
 - QwtKnob, 243
 - QwtSlider, 729

- QwtThermo, 782
- QwtWheel, 797
- setOrientations
 - QwtPanner, 315
 - QwtPlotPanner, 494
- setOrigin
 - QwtAnalogClock, 76
 - QwtCompass, 117
 - QwtDial, 172
- setPaintAttribute
 - QwtPlotCanvas, 389
 - QwtPlotCurve, 411
 - QwtText, 758
- setPaintInterval
 - QwtScaleMap, 695
- setPaintXInterval
 - QwtScaleMap, 695
- setPalette
 - QwtCompassMagnetNeedle, 129
 - QwtCompassRose, 131
 - QwtCompassWindArrow, 134
 - QwtDialNeedle, 183
 - QwtDialSimpleNeedle, 196
 - QwtPlotScaleItem, 565
 - QwtSimpleCompassRose, 712
- setPen
 - QwtPlotCurve, 411
 - QwtPlotGrid, 430
 - QwtSymbol, 749
- setPenWidth
 - QwtDialScaleDraw, 191
 - QwtScaleWidget, 706
- setPeriodic
 - QwtAbstractSlider, 47
 - QwtAnalogClock, 76
 - QwtCompass, 117
 - QwtCounter, 143
 - QwtDial, 173
 - QwtDoubleRange, 209
 - QwtKnob, 243
 - QwtSlider, 729
 - QwtWheel, 797
- setPipeWidth
 - QwtThermo, 782
- setPoints
 - QwtSpline, 741
- setPosition
 - QwtAbstractSlider, 47
 - QwtAnalogClock, 77
 - QwtCompass, 117
 - QwtDial, 173
 - QwtKnob, 244
 - QwtPlotScaleItem, 565
 - QwtSlider, 729
 - QwtThermo, 783
 - QwtWheel, 798
- setRadius
 - QwtDialScaleDraw, 191
 - QwtRoundScaleDraw, 661
- setRange
 - QwtAbstractSlider, 47
 - QwtAnalogClock, 77
 - QwtCompass, 118
 - QwtCounter, 143
 - QwtDial, 173
 - QwtDoubleRange, 209
 - QwtKnob, 244
 - QwtSlider, 729
 - QwtThermo, 783
 - QwtWheel, 798
- setRawData
 - QwtPlotCurve, 411
- setReadOnly
 - QwtAbstractSlider, 48
 - QwtAnalogClock, 77
 - QwtCompass, 118
 - QwtDial, 174
 - QwtKnob, 244
 - QwtSlider, 730
 - QwtWheel, 798
- setReference
 - QwtLinearScaleEngine, 283
 - QwtLog10ScaleEngine, 290
 - QwtScaleEngine, 691
- setReferenceAxis
 - QwtPlotRescaler, 554
- setRenderFlags
 - QwtText, 758
- setRenderHint
 - QwtPlotCurve, 412
 - QwtPlotGrid, 431
 - QwtPlotItem, 445
 - QwtPlotMarker, 480
 - QwtPlotRasterItem, 541
 - QwtPlotScaleItem, 566
 - QwtPlotSpectrogram, 588
 - QwtPlotSvgItem, 601
- setRescalePolicy
 - QwtPlotRescaler, 554
- setSizeMode
 - QwtPicker, 336

- QwtPlotPicker, 519
- QwtPlotZoomer, 632
- setRose
 - QwtCompass, 118
- setRubberBand
 - QwtPicker, 337
 - QwtPlotPicker, 519
 - QwtPlotZoomer, 632
- setRubberBandPen
 - QwtPicker, 337
 - QwtPlotPicker, 519
 - QwtPlotZoomer, 632
- setScale
 - QwtAbstractScale, 27, 28
 - QwtAnalogClock, 78
 - QwtCompass, 119
 - QwtDial, 174
 - QwtKnob, 245
 - QwtSlider, 730, 731
 - QwtThermo, 783, 784
- setScaleArc
 - QwtAnalogClock, 78
 - QwtCompass, 119
 - QwtDial, 174
- setScaleDiv
 - QwtAbstractScaleDraw, 36
 - QwtDialScaleDraw, 191
 - QwtPlotScaleItem, 566
 - QwtRoundScaleDraw, 661
 - QwtScaleDraw, 684
 - QwtScaleWidget, 706
- setScaleDivFromAxis
 - QwtPlotScaleItem, 566
- setScaleDraw
 - QwtAnalogClock, 78
 - QwtCompass, 119
 - QwtDial, 174
 - QwtKnob, 246
 - QwtPlotScaleItem, 567
 - QwtScaleWidget, 706
 - QwtSlider, 731
 - QwtThermo, 784
- setScaleEngine
 - QwtAbstractScale, 28
 - QwtKnob, 246
 - QwtSlider, 731
 - QwtThermo, 784
- setScaleInterval
 - QwtScaleMap, 695
- setScaleMaxMajor
 - QwtAbstractScale, 28
 - QwtKnob, 246
 - QwtSlider, 732
 - QwtThermo, 784
- setScaleMaxMinor
 - QwtAbstractScale, 29
 - QwtKnob, 246
 - QwtSlider, 732
 - QwtThermo, 785
- setScaleOptions
 - QwtAnalogClock, 78
 - QwtCompass, 119
 - QwtDial, 175
- setScalePosition
 - QwtSlider, 732
 - QwtThermo, 785
- setScaleTicks
 - QwtAnalogClock, 79
 - QwtCompass, 120
 - QwtDial, 175
- setScaleSelectionFlags
 - QwtPicker, 337
 - QwtPlotPicker, 520
 - QwtPlotZoomer, 633
- setSize
 - QwtSymbol, 749
- setSpacing
 - QwtAbstractScaleDraw, 36
 - QwtDialScaleDraw, 191
 - QwtLegendItem, 267
 - QwtPlotLayout, 458
 - QwtPlotMarker, 481
 - QwtRoundScaleDraw, 662
 - QwtScaleDraw, 684
 - QwtScaleWidget, 707
- setSplineSize
 - QwtSplineCurveFitter, 745
- setSplineType
 - QwtSpline, 742
- setState
 - QwtPickerClickPointMachine, 345
 - QwtPickerClickRectMachine, 347
 - QwtPickerDragPointMachine, 349
 - QwtPickerDragRectMachine, 351
 - QwtPickerMachine, 353
 - QwtPickerPolygonMachine, 355
- setStep
 - QwtAbstractSlider, 48
 - QwtAnalogClock, 79
 - QwtCompass, 120

- QwtCounter, [144](#)
- QwtDial, [175](#)
- QwtDoubleRange, [210](#)
- QwtKnob, [247](#)
- QwtSlider, [733](#)
- QwtWheel, [799](#)
- setStepButton1
 - QwtCounter, [144](#)
- setStepButton2
 - QwtCounter, [144](#)
- setStepButton3
 - QwtCounter, [144](#)
- setStyle
 - QwtPlotCurve, [412](#)
 - QwtSymbol, [749](#)
- setSymbol
 - QwtKnob, [247](#)
 - QwtLegendItem, [267](#)
 - QwtPlotCurve, [412](#)
 - QwtPlotMarker, [481](#)
- setText
 - QwtLegendItem, [268](#)
 - QwtText, [759](#)
 - QwtTextLabel, [769](#)
- setTextEngine
 - QwtText, [759](#)
- setThumbLength
 - QwtSlider, [733](#)
- setThumbWidth
 - QwtSlider, [733](#)
- setTickCnt
 - QwtWheel, [799](#)
- setTickLength
 - QwtAbstractScaleDraw, [36](#)
 - QwtDialScaleDraw, [192](#)
 - QwtRoundScaleDraw, [662](#)
 - QwtScaleDraw, [684](#)
- setTicks
 - QwtScaleDiv, [669](#)
- setTime
 - QwtAnalogClock, [79](#)
- setTitle
 - QwtPlot, [383](#)
 - QwtPlotCurve, [413](#)
 - QwtPlotGrid, [431](#)
 - QwtPlotItem, [446](#)
 - QwtPlotMarker, [481](#), [482](#)
 - QwtPlotRasterItem, [542](#)
 - QwtPlotScaleItem, [567](#)
 - QwtPlotSpectrogram, [588](#)
 - QwtPlotSvgItem, [602](#)
 - QwtScaleWidget, [707](#)
- setTotalAngle
 - QwtKnob, [247](#)
 - QwtWheel, [799](#)
- setTrackerFont
 - QwtPicker, [337](#)
 - QwtPlotPicker, [520](#)
 - QwtPlotZoomer, [633](#)
- setTrackerMode
 - QwtPicker, [338](#)
 - QwtPlotPicker, [520](#)
 - QwtPlotZoomer, [633](#)
- setTrackerPen
 - QwtPicker, [338](#)
 - QwtPlotPicker, [521](#)
 - QwtPlotZoomer, [634](#)
- setTracking
 - QwtAbstractSlider, [48](#)
 - QwtAnalogClock, [80](#)
 - QwtCompass, [120](#)
 - QwtDial, [176](#)
 - QwtKnob, [248](#)
 - QwtSlider, [734](#)
 - QwtWheel, [800](#)
- setTransformation
 - QwtAbstractScaleDraw, [36](#)
 - QwtDialScaleDraw, [192](#)
 - QwtRoundScaleDraw, [662](#)
 - QwtScaleDraw, [684](#)
 - QwtScaleMap, [695](#)
- setUpdateTime
 - QwtAbstractSlider, [49](#)
 - QwtAnalogClock, [80](#)
 - QwtCompass, [121](#)
 - QwtDial, [176](#)
 - QwtKnob, [248](#)
 - QwtSlider, [734](#)
 - QwtWheel, [800](#)
- setValid
 - QwtAbstractSlider, [49](#)
 - QwtAnalogClock, [80](#)
 - QwtCompass, [121](#)
 - QwtCounter, [145](#)
 - QwtDial, [176](#)
 - QwtDoubleRange, [210](#)
 - QwtKnob, [248](#)
 - QwtSlider, [734](#)
 - QwtWheel, [800](#)
- setValue

- QwtAbstractSlider, [49](#)
- QwtAnalogClock, [81](#)
- QwtCompass, [121](#)
- QwtCounter, [145](#)
- QwtDial, [177](#)
- QwtDoubleRange, [210](#)
- QwtKnob, [249](#)
- QwtPlotMarker, [482](#)
- QwtSlider, [735](#)
- QwtThermo, [786](#)
- QwtWheel, [801](#)
- setViewAngle
 - QwtWheel, [801](#)
- setVisible
 - QwtPlotCurve, [413](#)
 - QwtPlotGrid, [431](#)
 - QwtPlotItem, [446](#)
 - QwtPlotMarker, [482](#)
 - QwtPlotRasterItem, [542](#)
 - QwtPlotScaleItem, [568](#)
 - QwtPlotSpectrogram, [588](#)
 - QwtPlotSvgItem, [602](#)
- setWheelButtonState
 - QwtMagnifier, [297](#)
 - QwtPlotMagnifier, [466](#)
- setWheelFactor
 - QwtMagnifier, [297](#)
 - QwtPlotMagnifier, [466](#)
- setWheelWidth
 - QwtWheel, [801](#)
- setWidth
 - QwtDialSimpleNeedle, [196](#)
 - QwtSimpleCompassRose, [712](#)
- setWrapping
 - QwtAnalogClock, [81](#)
 - QwtCompass, [122](#)
 - QwtDial, [177](#)
- setXAxis
 - QwtPlotCurve, [414](#)
 - QwtPlotGrid, [432](#)
 - QwtPlotItem, [446](#)
 - QwtPlotMarker, [482](#)
 - QwtPlotRasterItem, [543](#)
 - QwtPlotScaleItem, [568](#)
 - QwtPlotSpectrogram, [589](#)
 - QwtPlotSvgItem, [602](#)
- setXDiv
 - QwtPlotGrid, [432](#)
- setXValue
 - QwtPlotMarker, [483](#)
- setYAxis
 - QwtPlotCurve, [414](#)
 - QwtPlotGrid, [432](#)
 - QwtPlotItem, [447](#)
 - QwtPlotMarker, [483](#)
 - QwtPlotRasterItem, [543](#)
 - QwtPlotScaleItem, [568](#)
 - QwtPlotSpectrogram, [589](#)
 - QwtPlotSvgItem, [603](#)
- setYDiv
 - QwtPlotGrid, [432](#)
- setYValue
 - QwtPlotMarker, [483](#)
- setZ
 - QwtPlotCurve, [414](#)
 - QwtPlotGrid, [433](#)
 - QwtPlotItem, [447](#)
 - QwtPlotMarker, [483](#)
 - QwtPlotRasterItem, [543](#)
 - QwtPlotScaleItem, [568](#)
 - QwtPlotSpectrogram, [589](#)
 - QwtPlotSvgItem, [603](#)
- setZoomBase
 - QwtPlotZoomer, [634](#), [635](#)
- setZoomInKey
 - QwtMagnifier, [297](#)
 - QwtPlotMagnifier, [466](#)
- setZoomOutKey
 - QwtMagnifier, [298](#)
 - QwtPlotMagnifier, [467](#)
- setZoomStack
 - QwtPlotZoomer, [635](#)
- Shadow
 - QwtAnalogClock, [61](#)
 - QwtCompass, [101](#)
 - QwtDial, [158](#)
- show
 - QwtPlotCurve, [414](#)
 - QwtPlotGrid, [433](#)
 - QwtPlotItem, [447](#)
 - QwtPlotMarker, [484](#)
 - QwtPlotRasterItem, [543](#)
 - QwtPlotScaleItem, [569](#)
 - QwtPlotSpectrogram, [590](#)
 - QwtPlotSvgItem, [603](#)
- showBackground
 - QwtAnalogClock, [81](#)
 - QwtCompass, [122](#)
 - QwtDial, [177](#)
- size

- QwtArrayData, 87
- QwtCPointerData, 148
- QwtData, 153
- QwtIntervalData, 229
- QwtPolygonFData, 645
- QwtSymbol, 750
- sizeHint
 - QwtAnalogClock, 82
 - QwtArrowButton, 91
 - QwtCompass, 123
 - QwtCounter, 145
 - QwtDial, 178
 - QwtDynGridLayout, 218
 - QwtKnob, 249
 - QwtLegend, 258
 - QwtLegendItem, 268
 - QwtPlot, 383
 - QwtScaleWidget, 708
 - QwtSlider, 735
 - QwtTextLabel, 769
 - QwtThermo, 786
 - QwtWheel, 802
- sliderMoved
 - QwtAbstractSlider, 49
 - QwtAnalogClock, 82
 - QwtCompass, 123
 - QwtDial, 178
 - QwtKnob, 249
 - QwtSlider, 735
 - QwtWheel, 802
- sliderPressed
 - QwtAbstractSlider, 50
 - QwtAnalogClock, 82
 - QwtCompass, 123
 - QwtDial, 178
 - QwtKnob, 249
 - QwtSlider, 735
 - QwtWheel, 802
- sliderReleased
 - QwtAbstractSlider, 50
 - QwtAnalogClock, 82
 - QwtCompass, 123
 - QwtDial, 178
 - QwtKnob, 249
 - QwtSlider, 735
 - QwtWheel, 802
- spacing
 - QwtAbstractScaleDraw, 37
 - QwtDialScaleDraw, 192
 - QwtLegendItem, 268
 - QwtPlotLayout, 459
 - QwtPlotMarker, 484
 - QwtRoundScaleDraw, 663
 - QwtScaleDraw, 685
 - QwtScaleWidget, 708
- splineSize
 - QwtSplineCurveFitter, 745
- SplineType
 - QwtSpline, 739
- splineType
 - QwtSpline, 742
- startBorderDist
 - QwtScaleWidget, 708
- state
 - QwtPickerClickPointMachine, 345
 - QwtPickerClickRectMachine, 347
 - QwtPickerDragPointMachine, 349
 - QwtPickerDragRectMachine, 351
 - QwtPickerMachine, 353
 - QwtPickerPolygonMachine, 355
- stateMachine
 - QwtPicker, 338
 - QwtPlotPicker, 521
 - QwtPlotZoomer, 635
- step
 - QwtAbstractSlider, 50
 - QwtAnalogClock, 82
 - QwtCompass, 123
 - QwtCounter, 145
 - QwtDial, 178
 - QwtDoubleRange, 211
 - QwtKnob, 250
 - QwtSlider, 736
 - QwtWheel, 802
- stepButton1
 - QwtCounter, 145
- stepButton2
 - QwtCounter, 145
- stepButton3
 - QwtCounter, 146
- stepChange
 - QwtAbstractSlider, 50
 - QwtAnalogClock, 83
 - QwtCompass, 123
 - QwtCounter, 146
 - QwtDial, 179
 - QwtDoubleRange, 211
 - QwtKnob, 250
 - QwtSlider, 736
 - QwtWheel, 802

- stopMoving
 - QwtAbstractSlider, 50
 - QwtAnalogClock, 83
 - QwtCompass, 124
 - QwtDial, 179
 - QwtKnob, 250
 - QwtSlider, 736
 - QwtWheel, 803
- stretchGrid
 - QwtDynGridLayout, 218
- stretchSelection
 - QwtPicker, 339
 - QwtPlotPicker, 522
 - QwtPlotZoomer, 636
- strip
 - QwtLinearScaleEngine, 283
 - QwtLog10ScaleEngine, 290
 - QwtScaleEngine, 691
- Style
 - QwtCompassMagnetNeedle, 127
 - QwtCompassWindArrow, 132
 - QwtDialSimpleNeedle, 194
 - QwtSymbol, 746
- style
 - QwtPlotCurve, 415
 - QwtSymbol, 750
- Symbol
 - QwtKnob, 233
- symbol
 - QwtKnob, 250
 - QwtLegendItem, 268
 - QwtPlotCurve, 415
 - QwtPlotMarker, 484
- symmetrize
 - QwtDoubleInterval, 204
- syncScale
 - QwtPlotRescaler, 554
- takeAt
 - QwtDynGridLayout, 218
- testAttribute
 - QwtLinearScaleEngine, 284
 - QwtLog10ScaleEngine, 291
 - QwtScaleEngine, 691
- testConrecAttribute
 - QwtPlotSpectrogram, 590
- testCurveAttribute
 - QwtPlotCurve, 415
- testDisplayMode
 - QwtPlotSpectrogram, 590
- testItemAttribute
 - QwtPlotCurve, 415
 - QwtPlotGrid, 433
 - QwtPlotItem, 447
 - QwtPlotMarker, 484
 - QwtPlotRasterItem, 544
 - QwtPlotScaleItem, 569
 - QwtPlotSpectrogram, 590
 - QwtPlotSvgItem, 603
- testLayoutAttribute
 - QwtText, 760
- testPaintAttribute
 - QwtPlotCanvas, 390
 - QwtPlotCurve, 416
 - QwtText, 760
- testRenderHint
 - QwtPlotCurve, 416
 - QwtPlotGrid, 433
 - QwtPlotItem, 448
 - QwtPlotMarker, 484
 - QwtPlotRasterItem, 544
 - QwtPlotScaleItem, 569
 - QwtPlotSpectrogram, 591
 - QwtPlotSvgItem, 604
- text
 - QwtLegendItem, 269
 - QwtText, 760
 - QwtTextLabel, 769
- textEngine
 - QwtText, 760, 761
- TextFormat
 - QwtText, 753
- textMargins
 - QwtMathMLTextEngine, 302
 - QwtPlainTextEngine, 358
 - QwtRichTextEngine, 652
 - QwtTextEngine, 764
- textRect
 - QwtLegendItem, 269
 - QwtTextLabel, 770
- textSize
 - QwtMathMLTextEngine, 303
 - QwtPlainTextEngine, 358
 - QwtRichTextEngine, 653
 - QwtText, 761
 - QwtTextEngine, 765
- thumbLength
 - QwtSlider, 736
- thumbWidth
 - QwtSlider, 736

- tickCnt
 - QwtWheel, [803](#)
- tickLabel
 - QwtAbstractScaleDraw, [37](#)
 - QwtDialScaleDraw, [192](#)
 - QwtRoundScaleDraw, [663](#)
 - QwtScaleDraw, [685](#)
- tickLength
 - QwtAbstractScaleDraw, [37](#)
 - QwtDialScaleDraw, [193](#)
 - QwtRoundScaleDraw, [663](#)
 - QwtScaleDraw, [685](#)
- ticks
 - QwtScaleDiv, [670](#)
- TickType
 - QwtScaleDiv, [667](#)
- timerEvent
 - QwtAbstractSlider, [50](#)
 - QwtAnalogClock, [83](#)
 - QwtCompass, [124](#)
 - QwtDial, [179](#)
 - QwtKnob, [250](#)
 - QwtSlider, [736](#)
 - QwtWheel, [803](#)
- title
 - QwtPlot, [383](#)
 - QwtPlotCurve, [416](#)
 - QwtPlotGrid, [434](#)
 - QwtPlotItem, [448](#)
 - QwtPlotMarker, [485](#)
 - QwtPlotRasterItem, [544](#)
 - QwtPlotScaleItem, [570](#)
 - QwtPlotSpectrogram, [591](#)
 - QwtPlotSvgItem, [604](#)
 - QwtScaleWidget, [708](#)
- titleHeightForWidth
 - QwtScaleWidget, [708](#)
- titleLabel
 - QwtPlot, [384](#)
- titleRect
 - QwtPlotLayout, [459](#)
- totalAngle
 - QwtKnob, [251](#)
 - QwtWheel, [803](#)
- trackerFont
 - QwtPicker, [339](#)
 - QwtPlotPicker, [522](#)
 - QwtPlotZoomer, [636](#)
- trackerMode
 - QwtPicker, [340](#)
 - QwtPlotPicker, [522](#)
 - QwtPlotZoomer, [636](#)
- trackerPen
 - QwtPicker, [340](#)
 - QwtPlotPicker, [522](#)
 - QwtPlotZoomer, [637](#)
- trackerPosition
 - QwtPicker, [340](#)
 - QwtPlotPicker, [523](#)
 - QwtPlotZoomer, [637](#)
- trackerRect
 - QwtPicker, [340](#)
 - QwtPlotPicker, [523](#)
 - QwtPlotZoomer, [637](#)
- trackerText
 - QwtPicker, [340](#)
 - QwtPlotPicker, [523, 524](#)
 - QwtPlotZoomer, [637, 638](#)
- trackerWidget
 - QwtPicker, [341](#)
 - QwtPlotPicker, [524](#)
 - QwtPlotZoomer, [638](#)
- transform
 - QwtPlot, [384](#)
 - QwtPlotCurve, [416](#)
 - QwtPlotGrid, [434](#)
 - QwtPlotItem, [448](#)
 - QwtPlotMarker, [485](#)
 - QwtPlotPicker, [524](#)
 - QwtPlotRasterItem, [544](#)
 - QwtPlotScaleItem, [570](#)
 - QwtPlotSpectrogram, [591](#)
 - QwtPlotSvgItem, [604](#)
 - QwtPlotZoomer, [638](#)
 - QwtScaleMap, [696](#)
- transformation
 - QwtLinearScaleEngine, [284](#)
 - QwtLog10ScaleEngine, [291](#)
 - QwtScaleEngine, [692](#)
 - QwtScaleMap, [696](#)
- transition
 - QwtPicker, [341](#)
 - QwtPickerClickPointMachine, [345](#)
 - QwtPickerClickRectMachine, [347](#)
 - QwtPickerDragPointMachine, [349](#)
 - QwtPickerDragRectMachine, [351](#)
 - QwtPickerMachine, [353](#)
 - QwtPickerPolygonMachine, [355](#)
 - QwtPlotPicker, [525](#)
 - QwtPlotZoomer, [639](#)

- translate
 - QwtMetricsMap, [304](#), [305](#)
- type
 - QwtScaleTransformation, [698](#)
- unite
 - QwtDoubleInterval, [204](#)
- updateAxes
 - QwtPlot, [384](#)
- updateDisplay
 - QwtPicker, [341](#)
 - QwtPlotPicker, [525](#)
 - QwtPlotZoomer, [639](#)
- updateLayout
 - QwtPlot, [384](#)
- updateLegend
 - QwtLegendItemManager, [271](#)
 - QwtPlotCurve, [417](#)
 - QwtPlotGrid, [434](#)
 - QwtPlotItem, [449](#)
 - QwtPlotMarker, [485](#)
 - QwtPlotRasterItem, [545](#)
 - QwtPlotScaleItem, [570](#)
 - QwtPlotSpectrogram, [592](#)
 - QwtPlotSvgItem, [605](#)
- updateMask
 - QwtAnalogClock, [83](#)
 - QwtCompass, [124](#)
 - QwtDial, [179](#)
- updateScale
 - QwtAnalogClock, [83](#)
 - QwtCompass, [124](#)
 - QwtDial, [179](#)
- updateScaleDiv
 - QwtPlotCurve, [417](#)
 - QwtPlotGrid, [435](#)
 - QwtPlotItem, [449](#)
 - QwtPlotMarker, [486](#)
 - QwtPlotRasterItem, [545](#)
 - QwtPlotScaleItem, [571](#)
 - QwtPlotSpectrogram, [592](#)
 - QwtPlotSvgItem, [605](#)
- updateScales
 - QwtPlotRescaler, [554](#)
- updateTabOrder
 - QwtPlot, [385](#)
- upperBound
 - QwtScaleDiv, [670](#)
- upperMargin
 - QwtLinearScaleEngine, [284](#)
 - QwtLog10ScaleEngine, [291](#)
 - QwtScaleEngine, [692](#)
- usedColor
 - QwtText, [762](#)
- usedFont
 - QwtText, [762](#)
- value
 - QwtAbstractSlider, [51](#)
 - QwtAnalogClock, [83](#)
 - QwtCompass, [124](#)
 - QwtCounter, [146](#)
 - QwtDial, [180](#)
 - QwtDoubleRange, [211](#)
 - QwtIntervalData, [229](#)
 - QwtKnob, [251](#)
 - QwtPlotMarker, [486](#)
 - QwtRasterData, [650](#)
 - QwtSlider, [737](#)
 - QwtSpline, [742](#)
 - QwtThermo, [786](#)
 - QwtWheel, [803](#)
- valueChange
 - QwtAbstractSlider, [51](#)
 - QwtAnalogClock, [84](#)
 - QwtCompass, [125](#)
 - QwtDial, [180](#)
 - QwtDoubleRange, [211](#)
 - QwtSlider, [737](#)
 - QwtWheel, [804](#)
- valueChanged
 - QwtAbstractSlider, [51](#)
 - QwtAnalogClock, [84](#)
 - QwtCompass, [125](#)
 - QwtCounter, [146](#)
 - QwtDial, [180](#)
 - QwtKnob, [251](#)
 - QwtSlider, [737](#)
 - QwtWheel, [804](#)
- verticalScrollBar
 - QwtLegend, [258](#)
- viewAngle
 - QwtWheel, [804](#)
- viewBox
 - QwtPlotSvgItem, [606](#)
- wheelButtonState
 - QwtMagnifier, [298](#)
 - QwtPlotMagnifier, [467](#)
- wheelEvent

- QwtAbstractSlider, [51](#)
- QwtAnalogClock, [84](#)
- QwtCompass, [125](#)
- QwtCounter, [146](#)
- QwtDial, [180](#)
- QwtKnob, [251](#)
- QwtSlider, [737](#)
- QwtWheel, [804](#)
- wheelFactor
 - QwtMagnifier, [298](#)
 - QwtPlotMagnifier, [467](#)
- widgetKeyPressEvent
 - QwtMagnifier, [298](#)
 - QwtPanner, [315](#)
 - QwtPicker, [341](#)
 - QwtPlotMagnifier, [468](#)
 - QwtPlotPanner, [494](#)
 - QwtPlotPicker, [525](#)
 - QwtPlotZoomer, [639](#)
- widgetKeyReleaseEvent
 - QwtMagnifier, [299](#)
 - QwtPanner, [315](#)
 - QwtPicker, [342](#)
 - QwtPlotMagnifier, [468](#)
 - QwtPlotPanner, [494](#)
 - QwtPlotPicker, [525](#)
 - QwtPlotZoomer, [639](#)
- widgetLeaveEvent
 - QwtPicker, [342](#)
 - QwtPlotPicker, [526](#)
 - QwtPlotZoomer, [640](#)
- widgetMouseDoubleClickEvent
 - QwtPicker, [342](#)
 - QwtPlotPicker, [526](#)
 - QwtPlotZoomer, [640](#)
- widgetMouseMoveEvent
 - QwtMagnifier, [299](#)
 - QwtPanner, [315](#)
 - QwtPicker, [343](#)
 - QwtPlotMagnifier, [468](#)
 - QwtPlotPanner, [495](#)
 - QwtPlotPicker, [526](#)
 - QwtPlotZoomer, [640](#)
- widgetMousePressEvent
 - QwtMagnifier, [299](#)
 - QwtPanner, [315](#)
 - QwtPicker, [343](#)
 - QwtPlotMagnifier, [468](#)
 - QwtPlotPanner, [495](#)
 - QwtPlotPicker, [526](#)
- QwtPlotZoomer, [640](#)
- widgetMouseReleaseEvent
 - QwtMagnifier, [299](#)
 - QwtPanner, [316](#)
 - QwtPicker, [343](#)
 - QwtPlotMagnifier, [469](#)
 - QwtPlotPanner, [495](#)
 - QwtPlotPicker, [526](#)
 - QwtPlotZoomer, [641](#)
- widgetWheelEvent
 - QwtMagnifier, [300](#)
 - QwtPicker, [343](#)
 - QwtPlotMagnifier, [469](#)
 - QwtPlotPicker, [527](#)
 - QwtPlotZoomer, [641](#)
- width
 - QwtDialSimpleNeedle, [197](#)
 - QwtDoubleInterval, [204](#)
 - QwtSimpleCompassRose, [712](#)
- wrapping
 - QwtAnalogClock, [84](#)
 - QwtCompass, [125](#)
 - QwtDial, [180](#)
- x
 - QwtArrayData, [87](#)
 - QwtCPointerData, [148](#)
 - QwtData, [153](#)
 - QwtPlotCurve, [417](#)
 - QwtPolygonFDData, [645](#)
- xAxis
 - QwtPlotCurve, [418](#)
 - QwtPlotGrid, [435](#)
 - QwtPlotItem, [449](#)
 - QwtPlotMarker, [486](#)
 - QwtPlotPicker, [527](#)
 - QwtPlotRasterItem, [546](#)
 - QwtPlotScaleItem, [571](#)
 - QwtPlotSpectrogram, [593](#)
 - QwtPlotSvgItem, [606](#)
 - QwtPlotZoomer, [641](#)
- xData
 - QwtArrayData, [87](#)
 - QwtCPointerData, [149](#)
- xEnabled
 - QwtPlotGrid, [435](#)
- xForm
 - QwtScaleTransformation, [698](#)
- xMinEnabled
 - QwtPlotGrid, [436](#)

- xScaleDiv
 - QwtPlotGrid, [436](#)
- xTransform
 - QwtScaleMap, [696](#)
- xValue
 - QwtPlotMarker, [487](#)
- xyPosition
 - QwtSlider, [737](#)
- y
 - QwtArrayData, [87](#)
 - QwtCPointerData, [149](#)
 - QwtData, [153](#)
 - QwtPlotCurve, [418](#)
 - QwtPolygonFData, [646](#)
- yAxis
 - QwtPlotCurve, [418](#)
 - QwtPlotGrid, [436](#)
 - QwtPlotItem, [450](#)
 - QwtPlotMarker, [487](#)
 - QwtPlotPicker, [527](#)
 - QwtPlotRasterItem, [546](#)
 - QwtPlotScaleItem, [571](#)
 - QwtPlotSpectrogram, [593](#)
 - QwtPlotSvgItem, [606](#)
 - QwtPlotZoomer, [641](#)
- yData
 - QwtArrayData, [88](#)
 - QwtCPointerData, [149](#)
- yEnabled
 - QwtPlotGrid, [436](#)
- yMinEnabled
 - QwtPlotGrid, [436](#)
- yScaleDiv
 - QwtPlotGrid, [437](#)
- yValue
 - QwtPlotMarker, [487](#)
- z
 - QwtPlotCurve, [418](#)
 - QwtPlotGrid, [437](#)
 - QwtPlotItem, [450](#)
 - QwtPlotMarker, [487](#)
 - QwtPlotRasterItem, [546](#)
 - QwtPlotScaleItem, [571](#)
 - QwtPlotSpectrogram, [593](#)
 - QwtPlotSvgItem, [606](#)
- zoom
 - QwtPlotZoomer, [642](#)
- zoomBase
 - QwtPlotZoomer, [642](#)
- zoomed
 - QwtPlotZoomer, [643](#)
- zoomRect
 - QwtPlotZoomer, [643](#)
- zoomRectIndex
 - QwtPlotZoomer, [643](#)
- zoomStack
 - QwtPlotZoomer, [643](#)