Qwt User's Guide

6.2.0

Generated by Doxygen 1.9.1

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 framework for 2D plots 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 6.2 might be usable in all environments where you find Qt. It is compatible with Qt 4.8 and all Qt5 versions.

1.3 What's new

Read the summary of the most important changes.

1.4 Screenshots

- Curve Plots
- Spectrogram, Contour Plot
- Bar Charts, Histograms
- Other Plots
- Dials, Compasses, Knobs, Wheels, Sliders, Thermos

1.5 Downloads

Stable releases or prereleases are available at the Qwt project page, or from the git repository.

Active development will happen in the develop branch supporting Qt >= 5.6 and relying on at least C++11. However all versions >= Qt 4.8 will actively be supported in the 6.x branches.
The popular multiaxes branch is derived from develop.

1.6 Installation

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

When no binary packages are available (f.e. on Windows) Qwt needs to be compiled and installed on the target system.

1.7 Support

- Mailing list
  For all kind of Qwt related questions use the Qwt mailing list.

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

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

1.8 Related Projects

PyQt-Qwt, Python PyQt wrapper for Qwt.

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)

2 What's new in Qwt 6.2

2.1 Qt >= 4.8

Support for Qt < Qt 4.8 has been dropped, support for Qt6 has been added.

2.2 Includes

Include files, that match the class names are available now. So it is possible to write "#include <QwtPlot>" now instead of "include qwt_plot.h"

2.3 QwtPolar has become part of Qwt

The QwtPolar project (https://qwtpolar.sourceforge.io) has been integrated into Qwt.

2.4 MathML Text Renderer has become its own project

The code can be found at https://github.com/uwerat/qwt-mml-dev now and is intended to become a standalone lib.

2.5 Spline

The broken implementation of QwtSpline has been replaced by a bunch of classes offering all sort of functionalities around splines.

The most popular spline approximation/interpolation algos have been implemented:

```
- Basis
- Cardinal
- ParabolicBlending
- Akima
- The one used in a proprietory office package
- Cubic
```

2.6 Better OpenGL support

QwtPlotOpenGLCanvas added to support rendering to a QOpenGLWidget

2.7 New plot items

```
- QwtPlotVectorField
  A new type of plot item for vector fields

- QwtPlotGraphicItem
  An item displaying a QwtGraphic image ( f.e used by QwtPlotSvgItem )
```

2.8 QwtPlotCurve

```
- QwtPlotCurve::FilterPointsAggressive
  A fast weeding algo for huge datasets with increasing x or y values

- QwtPlotCurve::closestPoint
  Is virtual now

- Line Clipping
  Includes the painter clip now

- QwtValuePointData
  A new type of data added, where the x values are the index

- QwtPlotCurve::setSamples
  more type of setters
```

2.9 QwtPlotSpectrogram

```
- QwtPlotSpectrogram::setColorTableSize
  Using individual RGB tables

- QwtRasterData::setInterval/interval
  API cleanup

- QwtHueColorMap, QwtSaturationValueColorMap
  New type of color maps

- QwtMatrixRasterData::BicubicInterpolation
  Implementation of bicubic interpolation added

- Handling of NaN values
  NaN values are interpreted as gaps, when QwtRasterData::WithoutGaps
  is not set
```

2.10 Other changes

```
- QwtPlotRenderer
  Using QPdfWriter instead of QPrinter, where possible

- Handling of Unknown Paint Engines
  Not aligning unknown paint engines ( f.e EMF ), QwtNullPaintDevice is not using
  QPaintEngine::User anymore

- QwtTransform::LogMin/LogMax
  LOG_MIN/LOG_MAX have been reomed ( values differ !
```

```
- qwt_compat.h
  Removed

- qwtFuzzyGreaterOrEqual/qwtFuzzyLessOrEqual
  Removed

- qwtGetMin/qwtGetMax
  Removed
```

3 Installing Qwt

3.1 Download

Stable Qwt releases are available from the Qwt project page.

Qwt-6.2.0 consists of 4 files:

- qwt-6.2.0.zip
  Zip file with the Qwt sources and the html documentation for Windows
- qwt-6.2.0.tar.bz2
  Compressed tar file with the Qwt sources and the html documentation for UNIX systems (Linux, Mac, ...)
- qwt-6.2.0.pdf
  Qwt documentation as PDF document.
- qwt-6.2.0.qch
  Qwt documentation as Qt Compressed Help document, that can be loaded into the Qt Assistant or Creator. In the Qt Creator context sensitive help will be available like for Qt classes.

Precompiled Qwt Designer plugins, that are compatible with some binary packages of the Qt Creator:

- qwtdesigner-6.2.0-∗.zip

3.2 Installing Qwt

Beside headers, libraries and the html version of the class documentation a proper Qwt installation contains a Designer plugin and a Qwt features file for building applications using Qwt.

All files will be copied to an installation directory, that is configurable by editing qwtconfig.pri. Its default settings is:

- Windows
  C:\Qwt-6.2.0
- Unix like systems
  /usr/local/qwt-6.2.0

For the rest of the document this install path will be written as *${QWT_ROOT}* and needs to be replaced by the real path in all commands below.

It is not unlikely, to have more than one installation of Qwt on the same system. F.e for using the Qwt Designer plugin in the Qt Creator a version of Qwt is necessary with the same Qt and compiler combination, that had been used for building the Qt Creator (see "Help->About Qt Creator ...").

Installing Qwt is done in 3 steps, that are quite common on UNIX systems.

1. Configuration
   In the configuration step all parameters are set to control how to build and install Qwt
2. Build
   In the build step binaries are built from the source files.
3. Installation
   The installation copies and rearranges all files that are necessary to build Qwt applications to a target directory.

The installation doesn't modify the system beside copying files to a directory in a proper way. After removing build and installation directories the system is in the same state as it was before.

3.2.1 Configuration

Configuring Qwt has to be done by editing the Project files used for building:

- qwtbuild.pri
  qwtbuild.pri contains settings for how to build Qwt. All settings of this file are only for building Qwt itself and doesn't have an impact on how an application using Qwt is built. Usually its default settings doesn't need to be modified.
- qwtconfig.pri
  qwtconfig.pri defines what modules of Qwt will be built and where to install them. qwtconfig.pri gets installed together with the Qwt features file qwt.prf and all its settings are known to project files for building Qwt applications.

In qwtconfig.pri the meaning of each option is explained in detail - it's worth reading it before running into problems later.

3.2.2 Build and installation

The Qt Creator is a graphical frontend for calling qmake/make and - technically - it could be used for building and installing Qwt. But as this way requires a lot more understanding of details the following step by step instructions are for the easier way using the command line.

3.2.2.1 Unix-like systems The first step before creating the Makefile is to check that the correct version of qmake is used. F.e. on older Linux distribution you often find a Qt3 qmake and in the path.

The default setting of qmake is to generate a makefile that builds Qwt for the same environment where the version of qmake has been built for. So creating a makefile usually means something like:

```
cd qwt-6.2.0
/usr/local/Qt-5.0.1/bin/qmake qwt.pro
```

The generated Makefile includes all paths related to the chosen Qt version and the next step is:

```
make
```

(On multicore systems you can speed up building the Qwt libraries with running several jobs simultaneously: f.e. "make -j4" on a dual core.)

Finally you have to install everything below the directories you have specified in qwtconfig.pri. Usually this is one of the system directories (/usr/local, /opt, ...) where you don't have write permission and then the installation needs to be done as root:

```
sudo make install
```

(On systems where sudo is not supported you can do the same with: su -c "make install")

3.2.2.2 Windows Qt packages offer a command line interface, that can be found in the Qt application menu: f.e "All Programs -> Qt -> Command Prompt". It is not mandatory to use it, but probably the easiest way as it offers an environment, where everything is initialized for a version of Qt (f.e qmake is in the PATH).

Creating a makefile usually means something like:
```
cd qwt-6.2.0
qmake qwt.pro
```

The generated makefile includes all paths related to the chosen Qt version.

3.2.2.2.1 MinGW For MinGW builds the name of the make tool is "mingw32-make"
```
mingw32-make
```

(On multicore systems you can speed up building the Qwt libraries with running several jobs simultaneously : "mingw32-make -j")

Finally you have to install everything below the directories you have specified in qwtconfig.pri.
```
mingw32-make install
```

3.2.2.2.2 MSVC For MSVC builds the name of the make tool is "nmake". Alternatively it is possible to use "jom" (https://wiki.qt.io/Jom), that is usually included in a Qt Creator package.
```
nmake
```

Finally you have to install everything below the directories you have specified in qwtconfig.pri.
```
nmake install
```

3.3 Qwt and the Qt tool chain

3.3.1 Designer plugin

The Designer plugin and the corresponding Qwt library (if the plugin has not been built self containing) have to be compatible with Qt version of the application loading it (usually the Qt Creator) - what is often a different version of the Qt libraries you want to build your application with. F.e on Windows the Qt Creator is usually built with a MSVC compiler - even if included in a MinGW package !

To help Qt Designer/Creator with locating the Qwt Designer plugin you have to set the environment variable QT_PLUGIN_PATH, modify qt.conf - or install the plugin to one of the application default paths.

The Qt documentation explains all options in detail:

- https://doc.qt.io/qt-5/deployment-plugins.html
- https://doc.qt.io/qtcreator/adding-plugins.html

F.e. on a Linux system you could add the following lines to .bashrc:
```
QT_PLUGIN_PATH="${QWT_ROOT}/plugins:$QT_PLUGIN_PATH"
export QT_PLUGIN_PATH
```

When the plugin has not been built including the Qwt library (see QwtDesignerSelfContained in qwtconfig.pri) the Qt Designer/Creator also needs to locate the Qwt libraries. On Unix systems the path to the installed library is compiled into the plugin (see rpath, ldd), but on Windows the Qt Creator needs to be configured ((Running a Qwt application) in the same way as for any application using Qwt.

In case of problems the diagnostics of Qt Creator and Designer are very limited (usually none), but setting the environment variable QT_DEBUG_PLUGINS might help. In the Qt Creator it is possible to check which plugins were loaded successfully and for certain problems it also lists those that were recognized but failed (*Tools* > *Form Editor* > *About Qt Designer Plugins*).

3.3.2 Online Help

The Qwt class documentation can be loaded into the Qt Creator:

- open the settings dialog from the *Tools->Options* menu
- raise the tab "Help->Documentation".
- press the *Add* button and select qwt-6.2.0.qch.

Now the context sensitive help (*F1*) works for Qwt classes.

For browsing the documentation in the Qt Assistant:

- open the settings dialog from the *Edit->Preferences* menu
- raise the tab *Documentation*.
- press the *Add* button and select qwt-6.2.0.qch.

3.4 Building a Qwt application

All flags and settings that are necessary to compile and link an application using Qwt can be found in the file ${QWT_ROOT}/features/qwt.prf.

When using qmake it can included from the application project file in 2 different ways:

- Adding Qwt as qmake feature

  When using the qmake feature mechanism you can bind a special version of qmake to a special installation of Qwt without having to add this dependency to the application project. How to add Qwt as feature is documented in the qmake docs.

  After adding Qwt as a feature f.e on Linux as a persistent property ....
  ```
  qmake -set QMAKEFEATURES ${QWT_ROOT}/features
  ```

  .. the following line can be added to the application project file:
  ```
  CONFIG += qwt
  ```

- Including qwt.prf in the application project file

  Instead of using qwt.prf as qmake feature it can be included from the application project file:

  ```
  include ( ${QWT_ROOT}/features/qwt.prf )
  ```

  The advantage of using a direct include is, that all settings of qwt.prf are known to the application project file (qmake features are included after the application project file has been parsed) and it can be implemented depending on - f.e. settings made in qwtconfig.pri.

On Unix platforms it is possible to link a runtime path into the executable, so that the location of the Qwt libraries can be found without having to configure a runtime environment:

- QMAKE_LFLAGS_RPATH
- QMAKE_RPATH
- QMAKE_RPATHDIR

3.5 Running a Qwt application

When using Qwt as shared library (DLL) the dynamic linker has to find it according to the rules of the operating system.

3.5.1 Windows

The only reasonable way to configure the runtime environment - without having to copy the Qwt libraries around - is to modify the PATH variable. F.e. this could be done by adding the following line to some batch file:

```
set PATH=%PATH%;${QWT_ROOT}\lib
```

3.5.2 GNU/Linux

Read the documentation about:

- *ldconfig*
- */etc/ld.so.conf*
- *LD_LIBRARY_PATH*

Using the *ldd* command a configuration can be tested.

4 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.

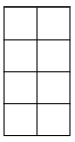
e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

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.

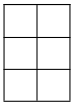
It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally

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

5 Curve Plots



6 Spectrogram, Contour Plot



7 Bar Charts, Histograms

8 Other Plots

9 Dials, Compasses, Knobs, Wheels, Sliders, Thermos

10 Namespace Index

10.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

11 Hierarchical Index

11.1 Class Hierarchy

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

12 Class Index

12.1 Class List

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

13 Namespace Documentation

13.1 QwtAxis Namespace Reference

Enumerations

- enum Position { YLeft , YRight , XBottom , XTop }
  Axis position.
- enum { **AxisPositions** = XTop + 1 }
  Number of axis positions.

Functions

- bool isValid (int axisPos)
- bool isYAxis (int axisPos)
- bool isXAxis (int axisPos)

13.1.1 Detailed Description

Enums and methods for axes

13.1.2 Enumeration Type Documentation

13.1.2.1 Position `enum QwtAxis::Position`

Axis position.

Enumerator

YLeft	Y axis left of the canvas.
YRight	Y axis right of the canvas.
XBottom	X axis below the canvas.
XTop	X axis above the canvas.

Definition at line 21 of file qwt_axis.h.

13.1.3 Function Documentation

13.1.3.1 isValid() `bool QwtAxis::isValid ( int axisPos ) [inline]`

Returns

true, when axisPos is in the valid range [YLeft, XTop]

Definition at line 45 of file qwt_axis.h.

13.1.3.2 isXAxis() `bool QwtAxis::isXAxis ( int axisPos ) [inline]`

Returns

true, when axisPos is XBottom or XTop

Definition at line 51 of file qwt_axis.h.

13.1.3.3 isYAxis() `bool QwtAxis::isYAxis ( int axisPos ) [inline]`

Returns

true, when axisPos is YLeft or YRight

Definition at line 57 of file qwt_axis.h.

13.2 QwtClipper Namespace Reference

Some clipping algorithms.

Functions

- QWT_EXPORT void clipPolygon (const QRect &, QPolygon &, bool closePolygon=false)
- QWT_EXPORT void clipPolygon (const QRectF &, QPolygon &, bool closePolygon=false)
- QWT_EXPORT void clipPolygonF (const QRectF &, QPolygonF &, bool closePolygon=false)
- QWT_EXPORT QPolygon clippedPolygon (const QRect &, const QPolygon &, bool closePolygon=false)
- QWT_EXPORT QPolygon clippedPolygon (const QRectF &, const QPolygon &, bool closePolygon=false)
- QWT_EXPORT QPolygonF clippedPolygonF (const QRectF &, const QPolygonF &, bool closePolygon=false)
- QWT_EXPORT QVector< QwtInterval > clipCircle (const QRectF &, const QPointF &, double radius)

13.2.1 Detailed Description

Some clipping algorithms.

13.2.2 Function Documentation

13.2.2.1 clipCircle() `QVector< QwtInterval > QwtClipper::clipCircle ( const QRectF & clipRect, const QPointF & center, double radius )`

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

clipRect	Clip rectangle
center	Center of the circle
radius	Radius of the circle

Returns

Arcs of the circle

Definition at line 477 of file qwt_clipper.cpp.

13.2.2.2 clippedPolygon() [1/2]

```
QPolygon QwtClipper::clippedPolygon (
        const QRect & clipRect,
        const QPolygon & polygon,
        bool closePolygon = false )
```

Sutherland-Hodgman polygon clipping

Parameters

clipRect	Clip rectangle
polygon	Polygon
closePolygon	True, when the polygon is closed

Returns

Clipped polygon

Definition at line 437 of file qwt_clipper.cpp.

13.2.2.3 clippedPolygon() [2/2]

```
QPolygon QwtClipper::clippedPolygon (
        const QRectF & clipRect,
        const QPolygon & polygon,
        bool closePolygon = false )
```

Sutherland-Hodgman polygon clipping

Parameters

clipRect	Clip rectangle
polygon	Polygon
closePolygon	True, when the polygon is closed

Returns

Clipped polygon

Definition at line 420 of file qwt_clipper.cpp.

13.2.2.4 clippedPolygonF()

```
QPolygonF QwtClipper::clippedPolygonF (
        const QRectF & clipRect,
        const QPolygonF & polygon,
        bool closePolygon = false )
```

Sutherland-Hodgman polygon clipping

Parameters

clipRect	Clip rectangle
polygon	Polygon
closePolygon	True, when the polygon is closed

Returns

Clipped polygon

Definition at line 455 of file qwt_clipper.cpp.

13.2.2.5 clipPolygon() [1/2]

```
void QwtClipper::clipPolygon (
        const QRect & clipRect,
        QPolygon & polygon,
        bool closePolygon = false )
```

Sutherland-Hodgman polygon clipping

Parameters

clipRect	Clip rectangle
polygon	Polygon IN/OUT
closePolygon	True, when the polygon is closed

Definition at line 390 of file qwt_clipper.cpp.

13.2.2.6 clipPolygon() [2/2]

```
void QwtClipper::clipPolygon (
        const QRectF & clipRect,
        QPolygon & polygon,
        bool closePolygon = false )
```

Sutherland-Hodgman polygon clipping

Parameters

clipRect	Clip rectangle
polygon	Polygon IN/OUT
closePolygon	True, when the polygon is closed

Definition at line 369 of file qwt_clipper.cpp.

13.2.2.7 clipPolygonF()

```
void QwtClipper::clipPolygonF (
        const QRectF & clipRect,
        QPolygonF & polygon,
        bool closePolygon = false )
```

Sutherland-Hodgman polygon clipping

Parameters

clipRect	Clip rectangle
polygon	Polygon IN/OUT
closePolygon	True, when the polygon is closed

Definition at line 404 of file qwt_clipper.cpp.

14 Class Documentation

14.1 QwtEventPattern::KeyPattern Class Reference

A pattern for key events.

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

Public Member Functions

- KeyPattern (int keyCode=Qt::Key_unknown, Qt::KeyboardModifiers modifierCodes=Qt::NoModifier)

  Constructor.

Public Attributes

- int key

  Key code.
- Qt::KeyboardModifiers modifiers

  Modifiers.

14.1.1 Detailed Description

A pattern for key events.

Definition at line 168 of file qwt_event_pattern.h.

14.2 QwtEventPattern::MousePattern Class Reference

A pattern for mouse events.

`#include <qwt_event_pattern.h>`

Public Member Functions

- MousePattern (Qt::MouseButton btn=Qt::NoButton, Qt::KeyboardModifiers modifierCodes=Qt::NoModifier)
  Constructor.

Public Attributes

- Qt::MouseButton button
  Button.
- Qt::KeyboardModifiers modifiers
  Keyboard modifier.

14.2.1 Detailed Description

A pattern for mouse events.

Definition at line 149 of file qwt_event_pattern.h.

14.3 QList< T > Class Template Reference

14.3.1 Detailed Description

template<typename T>
class QList< T >

Definition at line 17 of file qwt_abstract_legend.h.

14.4 QMap< Key, T > Class Template Reference

14.4.1 Detailed Description

template<class Key, class T>
class QMap< Key, T >

Definition at line 19 of file qwt_compass.h.

14.5 QStack< T > Class Template Reference

14.5.1 Detailed Description

template<typename T>
class QStack< T >

Definition at line 17 of file qwt_plot_zoomer.h.

14.6 QVector< T > Class Template Reference

14.6.1 Detailed Description

template<typename T>
class QVector< T >

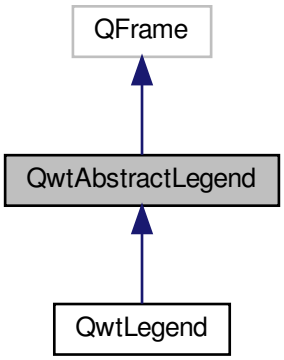
Definition at line 23 of file qwt_clipper.h.

14.7 QwtAbstractLegend Class Reference

Abstract base class for legend widgets.

`#include <qwt_abstract_legend.h>`

Inheritance diagram for QwtAbstractLegend:



Public Slots

- virtual void updateLegend (const QVariant &itemInfo, const QList< QwtLegendData > &data)=0

  Update the entries for a plot item.

Public Member Functions

- QwtAbstractLegend (QWidget ∗parent=NULL)
- virtual ∼QwtAbstractLegend ()

  Destructor.
- virtual void renderLegend (QPainter ∗painter, const QRectF &rect, bool fillBackground) const =0
- virtual bool isEmpty () const =0
- virtual int scrollExtent (Qt::Orientation) const

14.7.1 Detailed Description

Abstract base class for legend widgets.

Legends, that need to be under control of the QwtPlot layout system need to be derived from QwtAbstractLegend.

Note

Other type of legends can be implemented by connecting to the QwtPlot::legendDataChanged() signal. But as these legends are unknown to the plot layout system the layout code (on screen and for QwtPlotRenderer) need to be organized in application code.

See also

QwtLegend

Definition at line 34 of file qwt_abstract_legend.h.

14.7.2 Constructor & Destructor Documentation

14.7.2.1 QwtAbstractLegend() `QwtAbstractLegend::QwtAbstractLegend ( QWidget * parent = NULL ) [explicit]`

Constructor

Parameters

parent	Parent widget

Definition at line 18 of file qwt_abstract_legend.cpp.

14.7.3 Member Function Documentation

14.7.3.1 isEmpty() `virtual bool QwtAbstractLegend::isEmpty ( ) const [pure virtual]`

Returns

True, when no plot item is inserted

Implemented in QwtLegend.

14.7.3.2 renderLegend() `virtual void QwtAbstractLegend::renderLegend ( QPainter * painter, const QRectF & rect, bool fillBackground ) const [pure virtual]`

Render the legend into a given rectangle.

Parameters

painter	Painter
rect	Bounding rectangle
fillBackground	When true, fill rect with the widget background

See also

renderLegend() is used by QwtPlotRenderer

Implemented in QwtLegend.

14.7.3.3 scrollExtent() `int QwtAbstractLegend::scrollExtent ( Qt::Orientation orientation ) const [virtual]`

Return the extent, that is needed for elements to scroll the legend (usually scrollbars),

Parameters

orientation	Orientation

Returns

Extent of the corresponding scroll element

Reimplemented in QwtLegend.

Definition at line 35 of file qwt_abstract_legend.cpp.

14.7.3.4 updateLegend

```
virtual void QwtAbstractLegend::updateLegend (
        const QVariant & itemInfo,
        const QList< QwtLegendData > & data )  [pure virtual], [slot]
```

Update the entries for a plot item.

Parameters

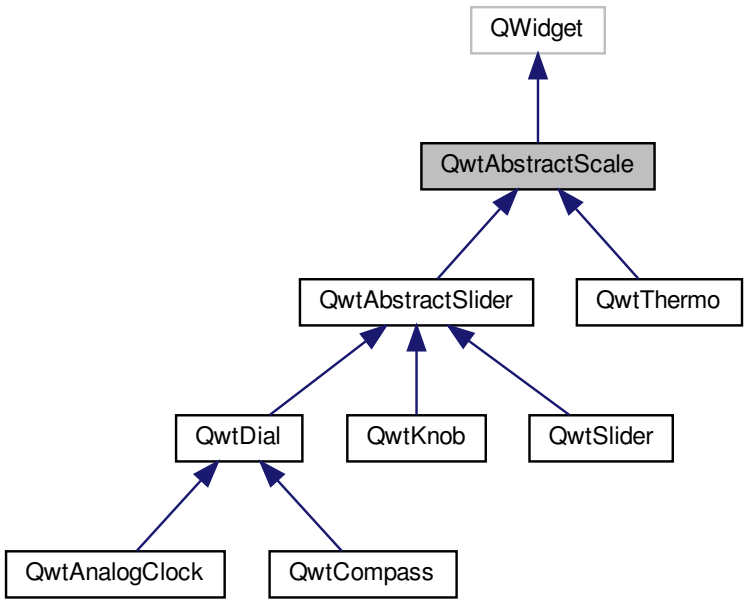
itemInfo	Info about an item
data	List of legend entry attributes for the item

14.8 QwtAbstractScale Class Reference

An abstract base class for widgets having a scale.

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

Inheritance diagram for QwtAbstractScale:



Public Member Functions

- QwtAbstractScale (QWidget ∗parent=NULL)
- virtual ∼QwtAbstractScale ()

  Destructor.
- void setScale (double lowerBound, double upperBound)

  Specify a scale.

- void setScale (const QwtInterval &)

  Specify a scale.
- void setScale (const QwtScaleDiv &)

  Specify a scale.
- const QwtScaleDiv & scaleDiv () const
- void setLowerBound (double value)
- double lowerBound () const
- void setUpperBound (double value)
- double upperBound () const
- void setScaleStepSize (double stepSize)

  Set the step size used for calculating a scale division.
- double scaleStepSize () const
- void setScaleMaxMajor (int ticks)

  Set the maximum number of major tick intervals.
- int scaleMaxMinor () const
- void setScaleMaxMinor (int ticks)

  Set the maximum number of minor tick intervals.
- int scaleMaxMajor () const
- void setScaleEngine (QwtScaleEngine *)

  Set a scale engine.
- const QwtScaleEngine * scaleEngine () const
- QwtScaleEngine * scaleEngine ()
- int transform (double) const
- double invTransform (int) const
- bool isInverted () const
- double minimum () const
- double maximum () const
- const QwtScaleMap & scaleMap () const

Protected Member Functions

- virtual void changeEvent (QEvent *) override
- void rescale (double lowerBound, double upperBound, double stepSize)
- void setAbstractScaleDraw (QwtAbstractScaleDraw *)

  Set a scale draw.
- const QwtAbstractScaleDraw * abstractScaleDraw () const
- QwtAbstractScaleDraw * abstractScaleDraw ()
- void updateScaleDraw ()
- virtual void scaleChange ()

  Notify changed scale.

14.8.1 Detailed Description

An abstract base class for widgets having a scale.

The scale of an QwtAbstractScale is determined by a QwtScaleDiv definition, that contains the boundaries and the ticks of the scale. The scale is painted using a QwtScaleDraw object.

The scale division might be assigned explicitly - but usually it is calculated from the boundaries using a QwtScaleEngine.

The scale engine also decides the type of transformation of the scale (linear, logarithmic ...).

Definition at line 36 of file qwt_abstract_scale.h.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 QwtAbstractScale() `QwtAbstractScale::QwtAbstractScale ( QWidget * parent = NULL ) [explicit]`

Constructor

Parameters

parent	Parent widget

Creates a default QwtScaleDraw and a QwtLinearScaleEngine. The initial scale boundaries are set to [0.0, 100.0]

The scaleStepSize() is initialized to 0.0, scaleMaxMajor() to 5 and scaleMaxMajor to 3.

Definition at line 57 of file qwt_abstract_scale.cpp.

14.8.3 Member Function Documentation

14.8.3.1 abstractScaleDraw() [1/2] `QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw ( ) [protected]`

Returns

Scale draw

See also

setAbstractScaleDraw()

Definition at line 293 of file qwt_abstract_scale.cpp.

14.8.3.2 abstractScaleDraw() [2/2] `const QwtAbstractScaleDraw * QwtAbstractScale::abstractScaleDraw ( ) const [protected]`

Returns

Scale draw

See also

setAbstractScaleDraw()

Definition at line 302 of file qwt_abstract_scale.cpp.

14.8.3.3 changeEvent() `void QwtAbstractScale::changeEvent ( QEvent * event ) [override], [protected], [virtual]`

Change Event handler

Parameters

event	Change event

Invalidates internal caches if necessary

Reimplemented in QwtThermo, QwtSlider, QwtKnob, and QwtDial.

Definition at line 453 of file qwt_abstract_scale.cpp.

14.8.3.4 invTransform() `double QwtAbstractScale::invTransform ( int value ) const`

Translate a widget coordinate into a scale value

Parameters

value	Widget coordinate

Returns

Corresponding scale coordinate for value

See also

scaleMap(), transform()

Definition at line 381 of file qwt_abstract_scale.cpp.

14.8.3.5 isInverted() `bool QwtAbstractScale::isInverted ( ) const`

Returns

True, when the scale is increasing in opposite direction to the widget coordinates

Definition at line 390 of file qwt_abstract_scale.cpp.

14.8.3.6 lowerBound() `double QwtAbstractScale::lowerBound ( ) const`

Returns

Lower bound of the scale

See also

setLowerBound(), setScale(), upperBound()

Definition at line 88 of file qwt_abstract_scale.cpp.

14.8.3.7 maximum() `double QwtAbstractScale::maximum ( ) const`

Returns

The boundary with the larger value

See also

minimum(), lowerBound(), upperBound()

Definition at line 409 of file qwt_abstract_scale.cpp.

14.8.3.8 minimum() `double QwtAbstractScale::minimum ( ) const`

Returns

The boundary with the smaller value

See also

maximum(), lowerBound(), upperBound()

Definition at line 399 of file qwt_abstract_scale.cpp.

14.8.3.9 rescale()

```
void QwtAbstractScale::rescale (
        double lowerBound,
        double upperBound,
        double stepSize ) [protected]
```

Recalculate the scale division and update the scale.

Parameters

lowerBound	Lower limit of the scale interval
upperBound	Upper limit of the scale interval
stepSize	Major step size

See also

scaleChange()

Definition at line 429 of file qwt_abstract_scale.cpp.

14.8.3.10 scaleDiv() const QwtScaleDiv & QwtAbstractScale::scaleDiv ( ) const

Returns

Scale boundaries and positions of the ticks

The scale division might have been assigned explicitly or calculated implicitly by rescale().

Definition at line 349 of file qwt_abstract_scale.cpp.

14.8.3.11 scaleEngine() [1/2] QwtScaleEngine * QwtAbstractScale::scaleEngine ( )

Returns

Scale engine

See also

setScaleEngine()

Definition at line 338 of file qwt_abstract_scale.cpp.

14.8.3.12 scaleEngine() [2/2] const QwtScaleEngine * QwtAbstractScale::scaleEngine ( ) const

Returns

Scale engine

See also

setScaleEngine()

Definition at line 329 of file qwt_abstract_scale.cpp.

14.8.3.13 scaleMap() const QwtScaleMap & QwtAbstractScale::scaleMap ( ) const

Returns

Map to translate between scale and widget coordinates

Definition at line 357 of file qwt_abstract_scale.cpp.

14.8.3.14 scaleMaxMajor() `int QwtAbstractScale::scaleMaxMajor ( ) const`

Returns

Maximal number of major tick intervals

See also

setScaleMaxMajor(), scaleMaxMinor()

Definition at line 202 of file qwt_abstract_scale.cpp.

14.8.3.15 scaleMaxMinor() `int QwtAbstractScale::scaleMaxMinor ( ) const`

Returns

Maximal number of minor tick intervals

See also

setScaleMaxMinor(), scaleMaxMajor()

Definition at line 232 of file qwt_abstract_scale.cpp.

14.8.3.16 scaleStepSize() `double QwtAbstractScale::scaleStepSize ( ) const`

Returns

Hint for the step size of the scale

See also

setScaleStepSize(), QwtScaleEngine::divideScale()

Definition at line 264 of file qwt_abstract_scale.cpp.

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

Set a scale draw.

scaleDraw has to be created with new and will be deleted in the destructor or the next call of setAbstractScaleDraw().

See also

abstractScaleDraw()

Definition at line 277 of file qwt_abstract_scale.cpp.

14.8.3.18 setLowerBound() `void QwtAbstractScale::setLowerBound ( double value )`

Set the lower bound of the scale

Parameters

value	Lower bound

See also

lowerBound(), setScale(), setUpperBound()

Note

For inverted scales the lower bound is greater than the upper bound

Definition at line 79 of file qwt_abstract_scale.cpp.

14.8.3.19 setScale() [1/3]

```
void QwtAbstractScale::setScale (
        const QwtInterval & interval )
```

Specify a scale.

Define a scale by an interval

The ticks are calculated using scaleMaxMinor(), scaleMaxMajor() and scaleStepSize().

Parameters

interval	Interval

Definition at line 145 of file qwt_abstract_scale.cpp.

14.8.3.20 setScale() [2/3]

```
void QwtAbstractScale::setScale (
        const QwtScaleDiv & scaleDiv )
```

Specify a scale.

scaleMaxMinor(), scaleMaxMajor() and scaleStepSize() and have no effect.

Parameters

scaleDiv	Scale division

See also

setAutoScale()

Definition at line 158 of file qwt_abstract_scale.cpp.

14.8.3.21 setScale() [3/3]

```
void QwtAbstractScale::setScale (
    double lowerBound,
    double upperBound )
```

Specify a scale.

Define a scale by an interval

The ticks are calculated using scaleMaxMinor(), scaleMaxMajor() and scaleStepSize().

Parameters

lowerBound	lower limit of the scale interval
upperBound	upper limit of the scale interval

Note

For inverted scales the lower bound is greater than the upper bound

Definition at line 130 of file qwt_abstract_scale.cpp.

14.8.3.22 setScaleEngine()

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

Set a scale engine.

The scale engine is responsible for calculating the scale division and provides a transformation between scale and widget coordinates.

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

Definition at line 316 of file qwt_abstract_scale.cpp.

14.8.3.23 setScaleMaxMajor()

```
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

ticks	Maximal number of major ticks.

See also

scaleMaxMajor(), setScaleMaxMinor(), setScaleStepSize(), QwtScaleEngine::divideInterval()

Definition at line 189 of file qwt_abstract_scale.cpp.

14.8.3.24 setScaleMaxMinor() `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

ticks	Maximal number of minor ticks.

See also

scaleMaxMajor(), setScaleMaxMinor(), setScaleStepSize(), QwtScaleEngine::divideInterval()

Definition at line 219 of file qwt_abstract_scale.cpp.

14.8.3.25 setScaleStepSize() `void QwtAbstractScale::setScaleStepSize ( double stepSize )`

Set the step size used for calculating a scale division.

The step size is hint for calculating the intervals for the major ticks of the scale. A value of 0.0 is interpreted as no hint.

Parameters

stepSize	Hint for the step size of the scale

See also

scaleStepSize(), QwtScaleEngine::divideScale()

Note

Position and distance between the major ticks also depends on scaleMaxMajor().

Definition at line 251 of file qwt_abstract_scale.cpp.

14.8.3.26 setUpperBound() `void QwtAbstractScale::setUpperBound ( double value )`

Set the upper bound of the scale

Parameters

value	Upper bound

See also

upperBound(), setScale(), setLowerBound()

Note

For inverted scales the lower bound is greater than the upper bound

Definition at line 102 of file qwt_abstract_scale.cpp.

14.8.3.27 transform() `int QwtAbstractScale::transform ( double value ) const`

Translate a scale value into a widget coordinate

Parameters

value	Scale value

Returns

Corresponding widget coordinate for value

See also

scaleMap(), invTransform()

Definition at line 369 of file qwt_abstract_scale.cpp.

14.8.3.28 updateScaleDraw() `void QwtAbstractScale::updateScaleDraw ( ) [protected]`

Recalculate ticks and scale boundaries.

Definition at line 466 of file qwt_abstract_scale.cpp.

14.8.3.29 upperBound() `double QwtAbstractScale::upperBound ( ) const`

Returns

Upper bound of the scale

See also

setUpperBound(), setScale(), lowerBound()

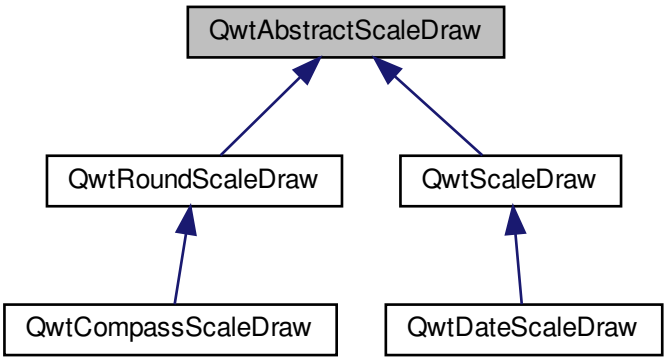
Definition at line 111 of file qwt_abstract_scale.cpp.

14.9 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 = 0x01 , Ticks = 0x02 , Labels = 0x04 }
- typedef QFlags< ScaleComponent > ScaleComponents

Public Member Functions

- QwtAbstractScaleDraw ()

  Constructor.
- virtual ~QwtAbstractScaleDraw ()

  Destructor.
- void setScaleDiv (const QwtScaleDiv &)
- const QwtScaleDiv & scaleDiv () const
- void setTransformation (QwtTransform *)
- const QwtScaleMap & scaleMap () const
- QwtScaleMap & scaleMap ()
- void enableComponent (ScaleComponent, bool enable=true)
- bool hasComponent (ScaleComponent) const
- void setTickLength (QwtScaleDiv::TickType, double length)
- double tickLength (QwtScaleDiv::TickType) const
- double maxTickLength () const
- void setSpacing (double)

  Set the spacing between tick and labels.
- double spacing () const

  Get the spacing.
- void setPenWidthF (qreal width)

  Specify the width of the scale pen.
- qreal penWidthF () const
- virtual void draw (QPainter *, const QPalette &) const

  Draw the scale.
- virtual QwtText label (double) const

  Convert a value into its representing label.
- virtual double extent (const QFont &font) const =0
- void setMinimumExtent (double)

  Set a minimum for the extent.
- double minimumExtent () const
- void invalidateCache ()

Protected Member Functions

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

  Convert a value into its representing label and cache it.

14.9.1 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 setScaleDiv(), the scale can be drawn with the draw() member.

Definition at line 31 of file qwt_abstract_scale_draw.h.

14.9.2 Member Typedef Documentation

14.9.2.1 ScaleComponents `typedef QFlags<ScaleComponent > QwtAbstractScaleDraw::ScaleComponents`

An ORed combination of ScaleComponent values.

Definition at line 51 of file qwt_abstract_scale_draw.h.

14.9.3 Member Enumeration Documentation

14.9.3.1 ScaleComponent `enum QwtAbstractScaleDraw::ScaleComponent`

Components of a scale

See also

enableComponent(), hasComponent

Enumerator

Backbone	Backbone = the line where the ticks are located.
Ticks	Ticks.
Labels	Labels.

Definition at line 39 of file qwt_abstract_scale_draw.h.

14.9.4 Constructor & Destructor Documentation

14.9.4.1 QwtAbstractScaleDraw() `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

Definition at line 60 of file qwt_abstract_scale_draw.cpp.

14.9.5 Member Function Documentation

14.9.5.1 draw()

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

Draw the scale.

Parameters

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

Definition at line 169 of file qwt_abstract_scale_draw.cpp.

14.9.5.2 drawBackbone()

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

Draws the baseline of the scale

Parameters

painter	Painter

See also

drawTick(), drawLabel()

Implemented in QwtScaleDraw, and QwtRoundScaleDraw.

14.9.5.3 drawLabel()

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

Draws the label for a major scale tick

Parameters

painter	Painter
value	Value

See also

drawTick(), drawBackbone()

Implemented in QwtScaleDraw, and QwtRoundScaleDraw.

14.9.5.4 drawTick()

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

Draw a tick

Parameters

painter	Painter
value	Value of the tick
len	Length of the tick

See also

drawBackbone(), drawLabel()

Implemented in QwtScaleDraw, and QwtRoundScaleDraw.

14.9.5.5 enableComponent()

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

En/Disable a component of the scale

Parameters

component	Scale component
enable	On/Off

See also

hasComponent()

Definition at line 79 of file qwt_abstract_scale_draw.cpp.

14.9.5.6 extent()

```
virtual double QwtAbstractScaleDraw::extent (
        const QFont & font ) 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.

Parameters

font	Font used for drawing the tick labels

Returns

Number of pixels

See also

setMinimumExtent(), minimumExtent()

Implemented in QwtScaleDraw, and QwtRoundScaleDraw.

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

Check if a component is enabled

Parameters

component	Component type

Returns

true, when component is enabled

See also

enableComponent()

Definition at line 95 of file qwt_abstract_scale_draw.cpp.

14.9.5.8 invalidateCache() `void QwtAbstractScaleDraw::invalidateCache ( )`

Invalidate the cache used by tickLabel()

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

Definition at line 417 of file qwt_abstract_scale_draw.cpp.

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

Convert a value into its representing label.

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

Parameters

value	Value

Returns

Label string.

Reimplemented in QwtDateScaleDraw, and QwtCompassScaleDraw.

Definition at line 375 of file qwt_abstract_scale_draw.cpp.

14.9.5.10 maxTickLength() `double QwtAbstractScaleDraw::maxTickLength ( ) const`

Returns

Length of the longest tick

Useful for layout calculations

See also

tickLength(), setTickLength()

Definition at line 355 of file qwt_abstract_scale_draw.cpp.

14.9.5.11 minimumExtent() `double QwtAbstractScaleDraw::minimumExtent ( ) const`

Get the minimum extent

Returns

Minimum extent

See also

extent(), setMinimumExtent()

Definition at line 302 of file qwt_abstract_scale_draw.cpp.

14.9.5.12 penWidthF() `qreal QwtAbstractScaleDraw::penWidthF ( ) const`

Returns

Scale pen width

See also

setPenWidth()

Definition at line 156 of file qwt_abstract_scale_draw.cpp.

14.9.5.13 scaleDiv() const QwtScaleDiv & QwtAbstractScaleDraw::scaleDiv ( ) const

Returns

scale division

Definition at line 133 of file qwt_abstract_scale_draw.cpp.

14.9.5.14 scaleMap() [1/2] QwtScaleMap & QwtAbstractScaleDraw::scaleMap ( )

Returns

Map how to translate between scale and pixel values

Definition at line 127 of file qwt_abstract_scale_draw.cpp.

14.9.5.15 scaleMap() [2/2] const QwtScaleMap & QwtAbstractScaleDraw::scaleMap ( ) const

Returns

Map how to translate between scale and pixel values

Definition at line 121 of file qwt_abstract_scale_draw.cpp.

14.9.5.16 setMinimumExtent() void QwtAbstractScaleDraw::setMinimumExtent (double *minExtent*)

Set a minimum for the extent.

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

Parameters

minExtent	Minimum extent

See also

extent(), minimumExtent()

Definition at line 289 of file qwt_abstract_scale_draw.cpp.

14.9.5.17 setPenWidthF() void QwtAbstractScaleDraw::setPenWidthF (qreal *width*)

Specify the width of the scale pen.

Parameters

width	Pen width

See also

penWidth()

Definition at line 144 of file qwt_abstract_scale_draw.cpp.

14.9.5.18 setScaleDiv() void QwtAbstractScaleDraw::setScaleDiv (const QwtScaleDiv & *scaleDiv*)

Change the scale division

Parameters

scaleDiv	New scale division

Definition at line 104 of file qwt_abstract_scale_draw.cpp.

14.9.5.19 setSpacing() void QwtAbstractScaleDraw::setSpacing (double *spacing*)

Set the spacing between tick and labels.

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

Parameters

spacing	Spacing

See also

spacing()

Definition at line 254 of file qwt_abstract_scale_draw.cpp.

14.9.5.20 setTickLength() `void QwtAbstractScaleDraw::setTickLength ( QwtScaleDiv::TickType tickType, double length )`

Set the length of the ticks

Parameters

tickType	Tick type
length	New length

Warning

the length is limited to [0..1000]

Definition at line 315 of file qwt_abstract_scale_draw.cpp.

14.9.5.21 setTransformation() `void QwtAbstractScaleDraw::setTransformation ( QwtTransform * transformation )`

Change the transformation of the scale

Parameters

transformation	New scale transformation

Definition at line 115 of file qwt_abstract_scale_draw.cpp.

14.9.5.22 spacing() `double QwtAbstractScaleDraw::spacing ( ) const`

Get the spacing.

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

Returns

Spacing

See also

setSpacing()

Definition at line 271 of file qwt_abstract_scale_draw.cpp.

14.9.5.23 tickLabel()

```
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

font	Font
value	Value

Returns

Tick label

Definition at line 393 of file qwt_abstract_scale_draw.cpp.

14.9.5.24 tickLength() `double QwtAbstractScaleDraw::tickLength ( QwtScaleDiv::TickType tickType ) const`

Returns

Length of the ticks

See also

setTickLength(), maxTickLength()

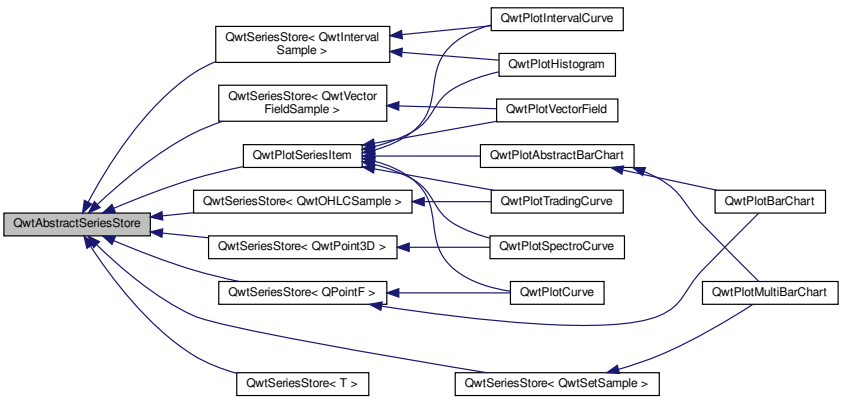
Definition at line 338 of file qwt_abstract_scale_draw.cpp.

14.10 QwtAbstractSeriesStore Class Reference

Bridge between QwtSeriesStore and QwtPlotSeriesItem.

`#include <qwt_series_store.h>`

Inheritance diagram for QwtAbstractSeriesStore:



Public Member Functions

- virtual ~QwtAbstractSeriesStore ()

  Destructor.

Protected Member Functions

- virtual void dataChanged ()=0

  dataChanged() indicates, that the series has been changed.
- virtual void setRectOfInterest (const QRectF &)=0
- virtual QRectF dataRect () const =0
- virtual size_t dataSize () const =0

14.10.1 Detailed Description

Bridge between QwtSeriesStore and QwtPlotSeriesItem.

QwtAbstractSeriesStore is an abstract interface only to make it possible to isolate the template based methods (QwtSeriesStore) from the regular methods (QwtPlotSeriesItem) to make it possible to derive from QwtPlotSeriesItem without any hassle with templates.

Definition at line 24 of file qwt_series_store.h.

14.10.2 Member Function Documentation

14.10.2.1 dataRect() `virtual QRectF QwtAbstractSeriesStore::dataRect ( ) const [protected], [pure virtual]`

Returns

Bounding rectangle of the stored series

Implemented in QwtSeriesStore< T >, QwtSeriesStore< QwtIntervalSample >, QwtSeriesStore< QwtVectorFieldSample >, QwtSeriesStore< QwtOHLCSample >, QwtSeriesStore< QPointF >, QwtSeriesStore< QwtPoint3D >, and QwtSeriesStore< QwtSetSample >.

14.10.2.2 dataSize() `virtual size_t QwtAbstractSeriesStore::dataSize ( ) const [protected], [pure virtual]`

Returns

Number of samples

Implemented in QwtSeriesStore< T >, QwtSeriesStore< QwtIntervalSample >, QwtSeriesStore< QwtVectorFieldSample >, QwtSeriesStore< QwtOHLCSample >, QwtSeriesStore< QPointF >, QwtSeriesStore< QwtPoint3D >, and QwtSeriesStore< QwtSetSample >.

14.10.2.3 setRectOfInterest() `virtual void QwtAbstractSeriesStore::setRectOfInterest ( const QRectF & ) [protected], [pure virtual]`

Set a the "rectangle of interest" for the stored series

See also

QwtSeriesData<T>::setRectOfInterest()

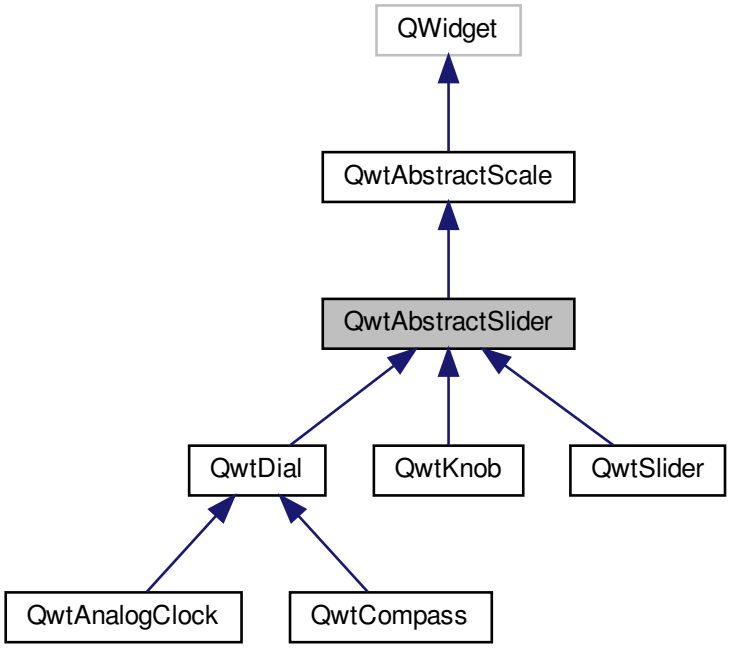
Implemented in QwtSeriesStore< T >, QwtSeriesStore< QwtIntervalSample >, QwtSeriesStore< QwtVectorFieldSample >, QwtSeriesStore< QwtOHLCSample >, QwtSeriesStore< QPointF >, QwtSeriesStore< QwtPoint3D >, and QwtSeriesStore< QwtSetSample >.

14.11 QwtAbstractSlider Class Reference

An abstract base class for slider widgets with a scale.

`#include <qwt_abstract_slider.h>`

Inheritance diagram for QwtAbstractSlider:



Public Slots

- void setValue (double value)

Signals

- void valueChanged (double value)

  Notify a change of value.
- void sliderPressed ()
- void sliderReleased ()
- void sliderMoved (double value)

Public Member Functions

- QwtAbstractSlider (QWidget *parent=NULL)

  Constructor.
- virtual ~QwtAbstractSlider ()

  Destructor.
- void setValid (bool)
- bool isValid () const
- double value () const

  Returns the current value.
- void setWrapping (bool)
- bool wrapping () const
- void setTotalSteps (uint)

  Set the number of steps.
- uint totalSteps () const
- void setSingleSteps (uint)

  Set the number of steps for a single increment.
- uint singleSteps () const
- void setPageSteps (uint)

  Set the number of steps for a page increment.
- uint pageSteps () const
- void setStepAlignment (bool)

  Enable step alignment.
- bool stepAlignment () const
- void setTracking (bool)

  Enables or disables tracking.
- bool isTracking () const
- void setReadOnly (bool)
- bool isReadOnly () const
- void setInvertedControls (bool)
- bool invertedControls () const

Protected Member Functions

- virtual void mousePressEvent (QMouseEvent *) override
- virtual void mouseReleaseEvent (QMouseEvent *) override
- virtual void mouseMoveEvent (QMouseEvent *) override
- virtual void keyPressEvent (QKeyEvent *) override
- virtual void wheelEvent (QWheelEvent *) override
- virtual bool isScrollPosition (const QPoint &pos) const =0

  Determine what to do when the user presses a mouse button.
- virtual double scrolledTo (const QPoint &pos) const =0

  Determine the value for a new position of the movable part of the slider.
- void incrementValue (int stepCount)
- virtual void scaleChange () override
- virtual void sliderChange ()

  Calling update()
- double incrementedValue (double value, int stepCount) const

14.11.1 Detailed Description

An abstract base class for slider widgets with a scale.

A slider widget displays a value according to a scale. The class is designed as a common super class for widgets like QwtKnob, QwtDial and QwtSlider.

When the slider is nor readOnly() its value can be modified by keyboard, mouse and wheel inputs.

The range of the slider is divided into a number of steps from which the value increments according to user inputs depend. Only for linear scales the number of steps correspond with a fixed step size.

Definition at line 32 of file qwt_abstract_slider.h.

14.11.2 Constructor & Destructor Documentation

14.11.2.1 QwtAbstractSlider() `QwtAbstractSlider::QwtAbstractSlider ( QWidget * parent = NULL ) [explicit]`

Constructor.

The scale is initialized to [0.0, 100.0], the number of steps is set to 100 with 1 and 10 and single an page step sizes. Step alignment is enabled.

The initial value is invalid.

Parameters

parent	Parent widget

Definition at line 91 of file qwt_abstract_slider.cpp.

14.11.3 Member Function Documentation

14.11.3.1 incrementedValue() `double QwtAbstractSlider::incrementedValue ( double value, int stepCount ) const [protected]`

Increment a value

Parameters

value	Value
stepCount	Number of steps

Returns

Incremented value

Definition at line 670 of file qwt_abstract_slider.cpp.

14.11.3.2 incrementValue() `void QwtAbstractSlider::incrementValue ( int stepCount ) [protected]`

Increment the slider

The step size depends on the number of totalSteps()

Parameters

stepCount	Number of steps

See also

setTotalSteps(), incrementedValue()

Definition at line 650 of file qwt_abstract_slider.cpp.

14.11.3.3 invertedControls() `bool QwtAbstractSlider::invertedControls ( ) const`

Returns

True, when the controls are inverted

See also

setInvertedControls()

Definition at line 637 of file qwt_abstract_slider.cpp.

14.11.3.4 isReadOnly() `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()

Definition at line 159 of file qwt_abstract_slider.cpp.

14.11.3.5 isScrollPosition() `virtual bool QwtAbstractSlider::isScrollPosition ( const QPoint & pos ) const [protected], [pure virtual]`

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

Parameters

pos	Mouse position

Return values

True,when	pos is a valid scroll position

See also

scrolledTo()

Implemented in QwtSlider, QwtKnob, and QwtDial.

14.11.3.6 isTracking() `bool QwtAbstractSlider::isTracking ( ) const`

Returns

True, when tracking has been enabled

See also

setTracking()

Definition at line 186 of file qwt_abstract_slider.cpp.

14.11.3.7 isValid() `bool QwtAbstractSlider::isValid ( ) const`

Returns

True, when the value is invalid

Definition at line 125 of file qwt_abstract_slider.cpp.

14.11.3.8 keyPressEvent() `void QwtAbstractSlider::keyPressEvent ( QKeyEvent * event ) [override], [protected], [virtual]`

Handles key events

QwtAbstractSlider handles the following keys:

- Qt::Key_Left
  Add/Subtract singleSteps() in direction to lowerBound();
- Qt::Key_Right
  Add/Subtract singleSteps() in direction to upperBound();
- Qt::Key_Down
  Subtract singleSteps(), when invertedControls() is false
- Qt::Key_Up
  Add singleSteps(), when invertedControls() is false
- Qt::Key_PageDown
  Subtract pageSteps(), when invertedControls() is false
- Qt::Key_PageUp
  Add pageSteps(), when invertedControls() is false
- Qt::Key_Home
  Set the value to the minimum()
- Qt::Key_End
  Set the value to the maximum()

Parameters

event	Key event

See also

isReadOnly()

Reimplemented in QwtCompass.

Definition at line 370 of file qwt_abstract_slider.cpp.

14.11.3.9 mouseMoveEvent() `void QwtAbstractSlider::mouseMoveEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse Move Event handler

Parameters

event	Mouse event

Definition at line 220 of file qwt_abstract_slider.cpp.

14.11.3.10 mousePressEvent() `void QwtAbstractSlider::mousePressEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse press event handler

Parameters

event	Mouse event

Reimplemented in QwtSlider.

Definition at line 195 of file qwt_abstract_slider.cpp.

14.11.3.11 mouseReleaseEvent() `void QwtAbstractSlider::mouseReleaseEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse Release Event handler

Parameters

event	Mouse event

Reimplemented in QwtSlider.

Definition at line 265 of file qwt_abstract_slider.cpp.

14.11.3.12 pageSteps() `uint QwtAbstractSlider::pageSteps ( ) const`

Returns

Number of steps

See also

setPageSteps(), totalSteps(), singleSteps()

Definition at line 533 of file qwt_abstract_slider.cpp.

14.11.3.13 scaleChange() `void QwtAbstractSlider::scaleChange ( ) [override], [protected], [virtual]`

Update the slider according to modifications of the scale

Reimplemented from QwtAbstractScale.

Reimplemented in QwtSlider, and QwtDial.

Definition at line 811 of file qwt_abstract_slider.cpp.

14.11.3.14 scrolledTo() `virtual double QwtAbstractSlider::scrolledTo ( const QPoint & pos ) const [protected], [pure virtual]`

Determine the value for a new position of the movable part of the slider.

Parameters

pos	Mouse position

Returns

Value for the mouse position

See also

isScrollPosition()

Implemented in QwtSlider, QwtKnob, and QwtDial.

14.11.3.15 setInvertedControls() `void QwtAbstractSlider::setInvertedControls ( bool on )`

Invert wheel and key events

Usually scrolling the mouse wheel "up" and using keys like page up will increase the slider's value towards its maximum. When invertedControls() is enabled the value is scrolled towards its minimum.

Inverting the controls might be f.e. useful for a vertical slider with an inverted scale (decreasing from top to bottom).

Parameters

on	Invert controls, when true

See also

invertedControls(), keyEvent(), wheelEvent()

Definition at line 628 of file qwt_abstract_slider.cpp.

14.11.3.16 setPageSteps() `void QwtAbstractSlider::setPageSteps ( uint stepCount )`

Set the number of steps for a page increment.

The range of the slider is divided into a number of steps from which the value increments according to user inputs depend.

Parameters

stepCount	Number of steps

See also

pageSteps(), setTotalSteps(), setSingleSteps()

Definition at line 524 of file qwt_abstract_slider.cpp.

14.11.3.17 setReadOnly() `void QwtAbstractSlider::setReadOnly ( bool on )`

En/Disable read only mode

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

Parameters

on	Enables in case of true

See also

isReadOnly()

Warning

The focus policy is set to Qt::StrongFocus or Qt::NoFocus

Definition at line 141 of file qwt_abstract_slider.cpp.

14.11.3.18 setSingleSteps() `void QwtAbstractSlider::setSingleSteps ( uint stepCount )`

Set the number of steps for a single increment.

The range of the slider is divided into a number of steps from which the value increments according to user inputs depend.

Parameters

stepCount	Number of steps

See also

singleSteps(), setTotalSteps(), setPageSteps()

Definition at line 499 of file qwt_abstract_slider.cpp.

14.11.3.19 setStepAlignment() `void QwtAbstractSlider::setStepAlignment ( bool on )`

Enable step alignment.

When step alignment is enabled values resulting from slider movements are aligned to the step size.

Parameters

on	Enable step alignment when true

See also

stepAlignment()

Definition at line 547 of file qwt_abstract_slider.cpp.

14.11.3.20 setTotalSteps() `void QwtAbstractSlider::setTotalSteps ( uint stepCount )`

Set the number of steps.

The range of the slider is divided into a number of steps from which the value increments according to user inputs depend.

The default setting is 100.

Parameters

stepCount	Number of steps

See also

totalSteps(), setSingleSteps(), setPageSteps()

Definition at line 474 of file qwt_abstract_slider.cpp.

14.11.3.21 setTracking() `void QwtAbstractSlider::setTracking ( bool on )`

Enables or disables tracking.

If tracking is enabled, the slider emits the valueChanged() signal while the movable part of the slider is being dragged. If tracking is disabled, the slider emits the valueChanged() signal only when the user releases the slider.

Tracking is enabled by default.

Parameters

on	`true` (enable) or `false` (disable) tracking.

See also

isTracking(), sliderMoved()

Definition at line 177 of file qwt_abstract_slider.cpp.

14.11.3.22 setValid() `void QwtAbstractSlider::setValid ( bool on )`

Set the value to be valid/invalid

Parameters

on	When true, the value is invalidated

See also

setValue()

Definition at line 113 of file qwt_abstract_slider.cpp.

14.11.3.23 setValue `void QwtAbstractSlider::setValue ( double value ) [slot]`

Set the slider to the specified value

Parameters

value	New value

See also

setValid(), sliderChange(), valueChanged()

Definition at line 570 of file qwt_abstract_slider.cpp.

14.11.3.24 setWrapping() `void QwtAbstractSlider::setWrapping ( bool on )`

If wrapping is true stepping up from upperBound() value will take you to the minimum() value and vice versa.

Parameters

on	En/Disable wrapping

See also

wrapping()

Definition at line 599 of file qwt_abstract_slider.cpp.

14.11.3.25 singleSteps() `uint QwtAbstractSlider::singleSteps ( ) const`

Returns

Number of steps

See also

setSingleSteps(), totalSteps(), pageSteps()

Definition at line 508 of file qwt_abstract_slider.cpp.

14.11.3.26 sliderMoved `void QwtAbstractSlider::sliderMoved ( double value ) [signal]`

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

Parameters

value	New value

See also

valueChanged()

14.11.3.27 sliderPressed `void QwtAbstractSlider::sliderPressed ( ) [signal]`

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

14.11.3.28 sliderReleased `void QwtAbstractSlider::sliderReleased ( ) [signal]`

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

14.11.3.29 stepAlignment() `bool QwtAbstractSlider::stepAlignment ( ) const`

Returns

True, when step alignment is enabled

See also

setStepAlignment()

Definition at line 559 of file qwt_abstract_slider.cpp.

14.11.3.30 totalSteps() `uint QwtAbstractSlider::totalSteps ( ) const`

Returns

Number of steps

See also

setTotalSteps(), singleSteps(), pageSteps()

Definition at line 483 of file qwt_abstract_slider.cpp.

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

Notify a change of value.

When tracking is enabled (default setting), this signal will be emitted every time the value changes.

Parameters

value	New value

See also

setTracking(), sliderMoved()

14.11.3.32 wheelEvent() `void QwtAbstractSlider::wheelEvent ( QWheelEvent * event ) [override], [protected], [virtual]`

Wheel Event handler

In/decreases the value by s number of steps. The direction depends on the invertedControls() property.

When the control or shift modifier is pressed the wheel delta (divided by 120) is mapped to an increment according to pageSteps(). Otherwise it is mapped to singleSteps().

Parameters

event	Wheel event

Reimplemented in QwtDial.

Definition at line 296 of file qwt_abstract_slider.cpp.

14.11.3.33 wrapping() `bool QwtAbstractSlider::wrapping ( ) const`

Returns

True, when wrapping is set

See also

setWrapping()

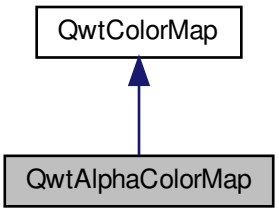
Definition at line 608 of file qwt_abstract_slider.cpp.

14.12 QwtAlphaColorMap Class Reference

QwtAlphaColorMap varies the alpha value of a color.

`#include <qwt_color_map.h>`

Inheritance diagram for QwtAlphaColorMap:



Public Member Functions

- QwtAlphaColorMap (const QColor &=QColor(Qt::gray))
  Constructor.
- virtual ~QwtAlphaColorMap ()
  Destructor.
- void setAlphaInterval (int alpha1, int alpha2)
- int alpha1 () const
- int alpha2 () const
- void setColor (const QColor &)
- QColor color () const
- virtual QRgb rgb (const QwtInterval &, double value) const override
  Map a value of a given interval into a alpha value.

Additional Inherited Members

14.12.1 Detailed Description

QwtAlphaColorMap varies the alpha value of a color.

Definition at line 147 of file qwt_color_map.h.

14.12.2 Constructor & Destructor Documentation

14.12.2.1 QwtAlphaColorMap() `QwtAlphaColorMap::QwtAlphaColorMap ( const QColor & color = QColor( Qt::gray ) ) [explicit]`

Constructor.

The alpha interval is initialized by 0 to 255.

Parameters

color	Color of the map

See also

setColor(), setAlphaInterval()

Definition at line 539 of file qwt_color_map.cpp.

14.12.3 Member Function Documentation

14.12.3.1 alpha1() `int QwtAlphaColorMap::alpha1 ( ) const`

Returns

First alpha coordinate

See also

setAlphaInterval()

Definition at line 600 of file qwt_color_map.cpp.

14.12.3.2 alpha2() `int QwtAlphaColorMap::alpha2 ( ) const`

Returns

Second alpha coordinate

See also

setAlphaInterval()

Definition at line 609 of file qwt_color_map.cpp.

14.12.3.3 color() `QColor QwtAlphaColorMap::color ( ) const`

Returns

the color

See also

setColor()

Definition at line 571 of file qwt_color_map.cpp.

14.12.3.4 rgb()

```
QRgb QwtAlphaColorMap::rgb (
        const QwtInterval & interval,
        double value ) const  [override], [virtual]
```

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

Parameters

interval	Range for all values
value	Value to map into a RGB value

Returns

RGB value, with an alpha value

Implements QwtColorMap.

Definition at line 622 of file qwt_color_map.cpp.

14.12.3.5 setAlphaInterval()

```
void QwtAlphaColorMap::setAlphaInterval (
    int alpha1,
    int alpha2 )
```

Set the interval for the alpha coordinate

alpha1/alpha2 need to be in the range 0 to 255, where 255 means opaque and 0 means transparent.

Parameters

alpha1	First alpha coordinate
alpha2	Second alpha coordinate

See also

alpha1(), alpha2()

Definition at line 587 of file qwt_color_map.cpp.

14.12.3.6 setColor()

```
void QwtAlphaColorMap::setColor (
    const QColor & color )
```

Set the color

Parameters

color	Color

See also

color()

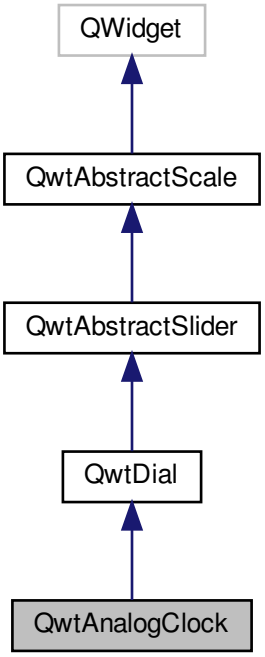
Definition at line 558 of file qwt_color_map.cpp.

14.13 QwtAnalogClock Class Reference

An analog clock.

`#include <qwt_analog_clock.h>`

Inheritance diagram for QwtAnalogClock:



Public Types

- enum Hand { SecondHand , MinuteHand , HourHand , NHands }

Public Slots

- void setCurrentTime ()

  Set the current time.
- void setTime (const QTime &)

Public Member Functions

- QwtAnalogClock (QWidget *parent=NULL)
- virtual ~QwtAnalogClock ()

  Destructor.
- void setHand (Hand, QwtDialNeedle *)
- const QwtDialNeedle * hand (Hand) const
- QwtDialNeedle * hand (Hand)

Protected Member Functions

- virtual void drawNeedle (QPainter *, const QPointF &, double radius, double direction, QPalette::ColorGroup) const override

  Draw the needle.
- virtual void drawHand (QPainter *, Hand, const QPointF &, double radius, double direction, QPalette::Color↩ Group) const

Additional Inherited Members

14.13.1 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);
```

Note

The examples/dials example shows how to use QwtAnalogClock.

Definition at line 43 of file qwt_analog_clock.h.

14.13.2 Member Enumeration Documentation

14.13.2.1 Hand `enum QwtAnalogClock::Hand`

Hand type

See also

setHand(), hand()

Enumerator

SecondHand	Needle displaying the seconds.
MinuteHand	Needle displaying the minutes.
HourHand	Needle displaying the hours.
NHands	Number of needles.

Definition at line 52 of file qwt_analog_clock.h.

14.13.3 Constructor & Destructor Documentation

14.13.3.1 QwtAnalogClock() `QwtAnalogClock::QwtAnalogClock ( QWidget * parent = NULL ) [explicit]`

Constructor

Parameters

parent	Parent widget

Definition at line 51 of file qwt_analog_clock.cpp.

14.13.4 Member Function Documentation

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

Draw a clock hand

Parameters

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

Definition at line 239 of file qwt_analog_clock.cpp.

14.13.4.2 drawNeedle() `void QwtAnalogClock::drawNeedle ( QPainter * painter, const QPointF & center,`

```
        double radius,
        double direction,
        QPalette::ColorGroup colorGroup ) const  [override], [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

painter	Painter
center	Center of the clock
radius	Maximum length for the hands
direction	Dummy, not used.
colorGroup	ColorGroup

See also

drawHand()

Reimplemented from QwtDial.

Definition at line 202 of file qwt_analog_clock.cpp.

14.13.4.3 hand() [1/2]

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

Returns

Clock hand

Parameters

hd	Specifies the type of hand

See also

setHand()

Definition at line 146 of file qwt_analog_clock.cpp.

14.13.4.4 hand() [2/2]

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

Returns

Clock hand

Parameters

hd	Specifies the type of hand

See also

setHand()

Definition at line 159 of file qwt_analog_clock.cpp.

14.13.4.5 setHand()

```
void QwtAnalogClock::setHand (
    Hand hand,
    QwtDialNeedle * needle )
```

Set a clock hand

Parameters

hand	Specifies the type of hand
needle	Hand

See also

hand()

Definition at line 132 of file qwt_analog_clock.cpp.

14.13.4.6 setTime

```
void QwtAnalogClock::setTime (
    const QTime & time )  [slot]
```

Set a time

Parameters

time	Time to display

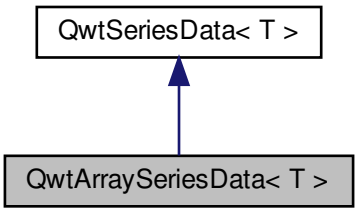
Definition at line 176 of file qwt_analog_clock.cpp.

14.14 QwtArraySeriesData< T > Class Template Reference

Template class for data, that is organized as QVector.

```
#include <qwt_series_data.h>
```

Inheritance diagram for QwtArraySeriesData< T >:



Public Member Functions

- QwtArraySeriesData ()
  Constructor.
- QwtArraySeriesData (const QVector< T > &samples)
- void setSamples (const QVector< T > &samples)
- const QVector< T > samples () const
- virtual size_t size () const override
- virtual T sample (size_t index) const override

Protected Attributes

- QVector< T > m_samples
  Vector of samples.

14.14.1 Detailed Description

template<typename T>
class QwtArraySeriesData< T >

Template class for data, that is organized as QVector.

QVector uses implicit data sharing and can be passed around as argument efficiently.

Definition at line 135 of file qwt_series_data.h.

14.14.2 Constructor & Destructor Documentation

14.14.2.1 QwtArraySeriesData() `template<typename T >`

```
QwtArraySeriesData< T >::QwtArraySeriesData (
            const QVector< T > & samples ) [explicit]
```

Constructor

Parameters

samples	Array of samples

Definition at line 178 of file qwt_series_data.h.

14.14.3 Member Function Documentation

14.14.3.1 sample() template<typename T >

```
T QwtArraySeriesData< T >::sample (
            size_t index ) const  [override], [virtual]
```

Returns

Sample at a specific position

Parameters

index	Index

Returns

Sample at position index

Implements QwtSeriesData< T >.

Reimplemented in QwtSyntheticPointData, QwtCPointerValueData< T >, QwtValuePointData< T >, QwtCPointerData< T >, and QwtPointArrayData< T >.

Definition at line 203 of file qwt_series_data.h.

14.14.3.2 samples() template<typename T >

```
const QVector< T > QwtArraySeriesData< T >::samples
```

Returns

Array of samples

Definition at line 191 of file qwt_series_data.h.

14.14.3.3 setSamples() template<typename T >

```
void QwtArraySeriesData< T >::setSamples (
            const QVector< T > & samples )
```

Assign an array of samples

Parameters

samples	Array of samples

Definition at line 184 of file qwt_series_data.h.

14.14.3.4 size() `template<typename T >`

`size_t QwtArraySeriesData< T >::size [override], [virtual]`

Returns

Number of samples

Implements QwtSeriesData< T >.

Reimplemented in QwtSyntheticPointData, QwtCPointerValueData< T >, QwtValuePointData< T >, QwtCPointerData< T >, and QwtPointArrayData< T >.

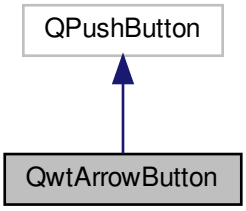
Definition at line 197 of file qwt_series_data.h.

14.15 QwtArrowButton Class Reference

Arrow Button.

`#include <qwt_arrow_button.h>`

Inheritance diagram for QwtArrowButton:



Public Member Functions

- QwtArrowButton (int num, Qt::ArrowType, QWidget *parent=NULL)
- virtual ~QwtArrowButton ()

  Destructor.
- Qt::ArrowType arrowType () const

  The direction of the arrows.
- int num () const

  The number of arrows.
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override

  Return a minimum size hint.

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗) override
- virtual void keyPressEvent (QKeyEvent ∗) override

  autoRepeat for the space keys
- virtual void drawButtonLabel (QPainter ∗)

  Draw the button label.
- virtual void drawArrow (QPainter ∗, const QRect &, Qt::ArrowType) const
- virtual QRect labelRect () const
- virtual QSize arrowSize (Qt::ArrowType, const QSize &boundingSize) const

14.15.1 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.

Definition at line 23 of file qwt_arrow_button.h.

14.15.2 Constructor & Destructor Documentation

14.15.2.1 QwtArrowButton()

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

Parameters

num	Number of arrows
arrowType	see Qt::ArrowType in the Qt docs.
parent	Parent widget

Definition at line 55 of file qwt_arrow_button.cpp.

14.15.3 Member Function Documentation

14.15.3.1 arrowSize()

```
QSize QwtArrowButton::arrowSize (
    Qt::ArrowType arrowType,
    const QSize & boundingSize ) const  [protected], [virtual]
```

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

Parameters

arrowType	Arrow type
boundingSize	Bounding size

Returns

Size of the arrow

Definition at line 296 of file qwt_arrow_button.cpp.

14.15.3.2 drawArrow()

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

Draw an arrow int a bounding rectangle

Parameters

painter	Painter
r	Rectangle where to paint the arrow
arrowType	Arrow type

Definition at line 215 of file qwt_arrow_button.cpp.

14.15.3.3 drawButtonLabel()

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

Draw the button label.

Parameters

painter	Painter

See also

The Qt Manual for QPushButton

Definition at line 143 of file qwt_arrow_button.cpp.

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

Returns

the bounding rectangle for the label

Definition at line 104 of file qwt_arrow_button.cpp.

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

Paint event handler

Parameters

event	Paint event

Definition at line 130 of file qwt_arrow_button.cpp.

14.15.3.6 sizeHint() `QSize QwtArrowButton::sizeHint ( ) const [override], [virtual]`

Returns

a size hint

Definition at line 259 of file qwt_arrow_button.cpp.

14.16 QwtBezier Class Reference

An implementation of the de Casteljau's Algorithm for interpolating Bézier curves.

`#include <qwt_bezier.h>`

Public Member Functions

- QwtBezier (double tolerance=0.5)

  Constructor.
- ~QwtBezier ()

  Destructor.
- void setTolerance (double tolerance)
- double tolerance () const
- QPolygonF toPolygon (const QPointF &p1, const QPointF &cp1, const QPointF &cp2, const QPointF &p2) const

  Interpolate a Bézier curve by a polygon.
- void appendToPolygon (const QPointF &p1, const QPointF &cp1, const QPointF &cp2, const QPointF &p2, QPolygonF &polygon) const

  Interpolate a Bézier curve by a polygon.

Static Public Member Functions

- static QPointF pointAt (const QPointF &p1, const QPointF &cp1, const QPointF &cp2, const QPointF &p2, double t)

14.16.1 Detailed Description

An implementation of the de Casteljau's Algorithm for interpolating Bézier curves.

The flatness criterion for terminating the subdivision is based on "Piecewise Linear Approximation of Bézier Curves" by Roger Willcocks (http://www.rops.org)

This article explains the maths behind in a very nice way: https://jeremykun.com/2013/05/11/bezier-curves-ar

Definition at line 29 of file qwt_bezier.h.

14.16.2 Constructor & Destructor Documentation

14.16.2.1 QwtBezier() QwtBezier::QwtBezier (double *tolerance* = *0.5*)

Constructor.

Parameters

tolerance	Termination criterion for the subdivision

See also

setTolerance()

Definition at line 116 of file qwt_bezier.cpp.

14.16.3 Member Function Documentation

14.16.3.1 appendToPolygon() void QwtBezier::appendToPolygon (const QPointF & *p1*, const QPointF & *cp1*, const QPointF & *cp2*, const QPointF & *p2*, QPolygonF & *polygon*) const

Interpolate a Bézier curve by a polygon.

appendToPolygon() is tailored for cumulating points from a sequence of bezier curves like being created by a spline interpolation.

Parameters

p1	Start point
cp1	First control point
cp2	Second control point
p2	End point
polygon	Polygon, where the interpolating points are added

Note

If the last point of the incoming polygon matches p1 it won't be inserted a second time.

Definition at line 186 of file qwt_bezier.cpp.

14.16.3.2 pointAt()

```
QPointF QwtBezier::pointAt (
    const QPointF & p1,
    const QPointF & cp1,
    const QPointF & cp2,
    const QPointF & p2,
    double t )  [static]
```

Find a point on a Bézier Curve

Parameters

p1	Start point
cp1	First control point
cp2	Second control point
p2	End point
t	Parameter value, something between [0,1]

Returns

Point on the curve

Definition at line 239 of file qwt_bezier.cpp.

14.16.3.3 setTolerance()

```
void QwtBezier::setTolerance (
    double tolerance )
```

Set the tolerance

The tolerance is a measurement for the flatness of a curve. A curve with a flatness below the tolerance is considered as being flat terminating the subdivision algorithm.

When interpolating a Bezier curve to render it as a sequence of lines to some sort of raster (f.e to screen) a value of 0.5 of the pixel size is a good value for the tolerance.

Parameters

tolerance	Termination criterion for the subdivision

See also

tolerance()

Definition at line 141 of file qwt_bezier.cpp.

14.16.3.4 tolerance() `double QwtBezier::tolerance ( ) const [inline]`

Returns

Tolerance, that is used as criterion for the subdivision

See also

setTolerance()

Definition at line 56 of file qwt_bezier.h.

14.16.3.5 toPolygon() `QPolygonF QwtBezier::toPolygon (`

```
const QPointF & p1,
const QPointF & cp1,
const QPointF & cp2,
const QPointF & p2 ) const
```

Interpolate a Bézier curve by a polygon.

Parameters

p1	Start point
cp1	First control point
cp2	Second control point
p2	End point

Returns

Interpolating polygon

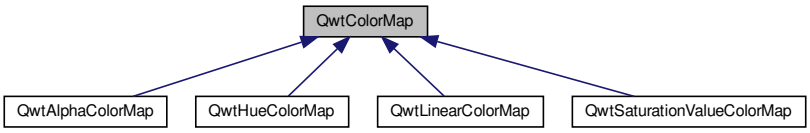
Definition at line 157 of file qwt_bezier.cpp.

14.17 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

- QwtColorMap (Format=QwtColorMap::RGB)
- virtual ~QwtColorMap ()

  Destructor.
- void setFormat (Format)
- virtual QRgb rgb (const QwtInterval &interval, double value) const =0
- virtual uint colorIndex (int numColors, const QwtInterval &interval, double value) const

  Map a value of a given interval into a color index.
- QColor color (const QwtInterval &, double value) const
- virtual QVector< QRgb > colorTable (int numColors) const
- virtual QVector< QRgb > colorTable256 () const

Public Attributes

- Format const

14.17.1 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

Definition at line 37 of file qwt_color_map.h.

14.17.2 Member Enumeration Documentation

14.17.2.1 Format `enum QwtColorMap::Format`

Format for color mapping

See also

rgb(), colorIndex(), colorTable()

Enumerator

RGB	The map is intended to map into RGB values.
Indexed	Map values into 8 bit values, that are used as indexes into the color table. Indexed color maps are used to generate QImage::Format_Indexed8 images. The calculation of the color index is usually faster and the resulting image has a lower memory footprint. **See also** colorIndex(), colorTable()

Definition at line 45 of file qwt_color_map.h.

14.17.3 Constructor & Destructor Documentation

14.17.3.1 QwtColorMap() `QwtColorMap::QwtColorMap ( Format format = QwtColorMap::RGB ) [explicit]`

Constructor

Parameters

format	Format of the color map

Definition at line 249 of file qwt_color_map.cpp.

14.17.4 Member Function Documentation

14.17.4.1 color() `QColor QwtColorMap::color ( const QwtInterval & interval, double value ) const [inline]`

Map a value into a color

Parameters

interval	Valid interval for values
value	Value

Returns

Color corresponding to value

Definition at line 248 of file qwt_color_map.h.

14.17.4.2 colorIndex()

```
uint QwtColorMap::colorIndex (
    int numColors,
    const QwtInterval & interval,
    double value ) const  [virtual]
```

Map a value of a given interval into a color index.

Parameters

numColors	Number of colors
interval	Range for all values
value	Value to map into a color index

Returns

Index, between 0 and numColors - 1, or -1 for an invalid value

Reimplemented in QwtLinearColorMap.

Definition at line 278 of file qwt_color_map.cpp.

14.17.4.3 colorTable()

```
QVector< QRgb > QwtColorMap::colorTable (
    int numColors ) const  [virtual]
```

Build and return a color map of arbitrary number of colors

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

Parameters

numColors	Number of colors

Returns

A color table

Definition at line 325 of file qwt_color_map.cpp.

14.17.4.4 colorTable256() `QVector< QRgb > QwtColorMap::colorTable256 ( ) 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().

Returns

A color table, that can be used for a QImage

Definition at line 304 of file qwt_color_map.cpp.

14.17.4.5 rgb()

```
virtual QRgb QwtColorMap::rgb (
    const QwtInterval & interval,
    double value ) const [pure virtual]
```

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

Parameters

interval	Range for the values
value	Value

Returns

RGB value, corresponding to value

Implemented in QwtSaturationValueColorMap, QwtHueColorMap, QwtAlphaColorMap, and QwtLinearColorMap.

14.17.4.6 setFormat()

```
void QwtColorMap::setFormat (
    Format format )
```

Set the format of the color map

Parameters

format	Format of the color map

Definition at line 264 of file qwt_color_map.cpp.

14.17.5 Member Data Documentation

14.17.5.1 const QwtColorMap::Format QwtColorMap::const [inline]

Initial value:

```
{
    return m_format
```

Returns

Intended format of the color map

See also

Format

Definition at line 67 of file qwt_color_map.h.

14.18 QwtColumnRect Class Reference

Directed rectangle representing bounding rectangle and orientation of a column.

`#include <qwt_column_symbol.h>`

Public Types

- enum Direction { LeftToRight , RightToLeft , BottomToTop , TopToBottom }
  Direction of the column.

Public Member Functions

- QwtColumnRect ()
  Build an rectangle with invalid intervals directed BottomToTop.
- QRectF toRect () const
- Qt::Orientation orientation () const

Public Attributes

- QwtInterval hInterval
  Interval for the horizontal coordinates.
- QwtInterval vInterval
  Interval for the vertical coordinates.
- Direction direction
  Direction.

14.18.1 Detailed Description

Directed rectangle representing bounding rectangle and orientation of a column.

Definition at line 26 of file qwt_column_symbol.h.

14.18.2 Member Enumeration Documentation

14.18.2.1 Direction `enum QwtColumnRect::Direction`

Direction of the column.

Enumerator

LeftToRight	From left to right.
RightToLeft	From right to left.
BottomToTop	From bottom to top.
TopToBottom	From top to bottom.

Definition at line 30 of file qwt_column_symbol.h.

14.18.3 Member Function Documentation

14.18.3.1 orientation() `Qt::Orientation QwtColumnRect::orientation ( ) const [inline]`

Returns

Orientation

Definition at line 55 of file qwt_column_symbol.h.

14.18.3.2 toRect() `QRectF QwtColumnRect::toRect ( ) const`

Returns

A normalized QRect built from the intervals

Definition at line 296 of file qwt_column_symbol.cpp.

14.19 QwtColumnSymbol Class Reference

A drawing primitive for columns.

`#include <qwt_column_symbol.h>`

Public Types

- enum Style { NoStyle = -1 , Box , UserStyle = 1000 }
- enum FrameStyle { NoFrame , Plain , Raised }

Public Member Functions

- QwtColumnSymbol (Style=NoStyle)
- virtual ~QwtColumnSymbol ()

  Destructor.
- void setFrameStyle (FrameStyle)
- FrameStyle frameStyle () const
- void setLineWidth (int width)
- int lineWidth () const
- void setPalette (const QPalette &)
- const QPalette & palette () const
- void setStyle (Style)
- Style style () const
- virtual void draw (QPainter *, const QwtColumnRect &) const

Protected Member Functions

- void drawBox (QPainter *, const QwtColumnRect &) const

14.19.1 Detailed Description

A drawing primitive for columns.

Definition at line 74 of file qwt_column_symbol.h.

14.19.2 Member Enumeration Documentation

14.19.2.1 FrameStyle `enum QwtColumnSymbol::FrameStyle`

Frame Style used in Box style().

See also

Style, setFrameStyle(), frameStyle(), setStyle(), setPalette()

Enumerator

NoFrame	No frame.
Plain	A plain frame style.
Raised	A raised frame style.

Definition at line 104 of file qwt_column_symbol.h.

14.19.2.2 Style `enum QwtColumnSymbol::Style`

Style

See also

setStyle(), style()

Enumerator

NoStyle	No Style, the symbol draws nothing.
Box	The column is painted with a frame depending on the frameStyle() and lineWidth() using the palette().
UserStyle	Styles >= QwtColumnSymbol::UserStyle are reserved for derived classes of QwtColumnSymbol that overload draw() with additional application specific symbol types.

Definition at line 81 of file qwt_column_symbol.h.

14.19.3 Constructor & Destructor Documentation

14.19.3.1 QwtColumnSymbol() `QwtColumnSymbol::QwtColumnSymbol ( Style style = NoStyle ) [explicit]`

Constructor

Parameters

style	Style of the symbol

See also

setStyle(), style(), Style

Definition at line 135 of file qwt_column_symbol.cpp.

14.19.4 Member Function Documentation

14.19.4.1 draw()

```
void QwtColumnSymbol::draw (
    QPainter * painter,
    const QwtColumnRect & rect ) const  [virtual]
```

Draw the symbol depending on its style.

Parameters

painter	Painter
rect	Directed rectangle

See also

drawBox()

Definition at line 238 of file qwt_column_symbol.cpp.

14.19.4.2 drawBox()

```
void QwtColumnSymbol::drawBox (
    QPainter * painter,
    const QwtColumnRect & rect ) const  [protected]
```

Draw the symbol when it is in Box style.

Parameters

painter	Painter
rect	Directed rectangle

See also

draw()

Definition at line 264 of file qwt_column_symbol.cpp.

14.19.4.3 frameStyle()

```
QwtColumnSymbol::FrameStyle QwtColumnSymbol::frameStyle ( ) const
```

Returns

Current frame style, that is used for the Box style.

See also

setFrameStyle(), lineWidth(), setStyle()

Definition at line 202 of file qwt_column_symbol.cpp.

14.19.4.4 lineWidth() `int QwtColumnSymbol::lineWidth ( ) const`

Returns

Line width of the frame, that is used for the Box style.

See also

setLineWidth(), frameStyle(), setStyle()

Definition at line 225 of file qwt_column_symbol.cpp.

14.19.4.5 palette() `const QPalette & QwtColumnSymbol::palette ( ) const`

Returns

Current palette

See also

setPalette()

Definition at line 182 of file qwt_column_symbol.cpp.

14.19.4.6 setFrameStyle() `void QwtColumnSymbol::setFrameStyle ( FrameStyle frameStyle )`

Set the frame, that is used for the Box style.

Parameters

frameStyle	Frame style

See also

frameStyle(), setLineWidth(), setStyle()

Definition at line 193 of file qwt_column_symbol.cpp.

14.19.4.7 setLineWidth() `void QwtColumnSymbol::setLineWidth ( int width )`

Set the line width of the frame, that is used for the Box style.

Parameters

width	Width

See also

lineWidth(), setFrameStyle()

Definition at line 213 of file qwt_column_symbol.cpp.

14.19.4.8 setPalette() `void QwtColumnSymbol::setPalette ( const QPalette & palette )`

Assign a palette for the symbol

Parameters

palette	Palette

See also

palette(), setStyle()

Definition at line 173 of file qwt_column_symbol.cpp.

14.19.4.9 setStyle() `void QwtColumnSymbol::setStyle ( Style style )`

Specify the symbol style

Parameters

style	Style

See also

style(), setPalette()

Definition at line 153 of file qwt_column_symbol.cpp.

14.19.4.10 style() `QwtColumnSymbol::Style QwtColumnSymbol::style ( ) const`

Returns

Current symbol style

See also

setStyle()

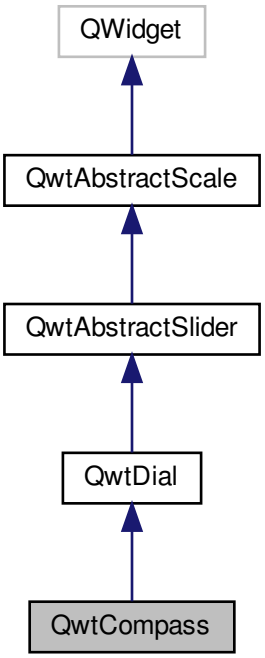
Definition at line 162 of file qwt_column_symbol.cpp.

14.20 QwtCompass Class Reference

A Compass Widget.

`#include <qwt_compass.h>`

Inheritance diagram for QwtCompass:



Public Member Functions

- QwtCompass (QWidget *parent=NULL)
  Constructor.
- virtual ~QwtCompass ()
  Destructor.
- void setRose (QwtCompassRose *rose)
- const QwtCompassRose * rose () const
- QwtCompassRose * rose ()

Protected Member Functions

- virtual void drawRose (QPainter *, const QPointF &center, double radius, double north, QPalette::Color↩ Group) const
- virtual void drawScaleContents (QPainter *, const QPointF &center, double radius) const override
- virtual void keyPressEvent (QKeyEvent *) override

Additional Inherited Members

14.20.1 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.

Definition at line 61 of file qwt_compass.h.

14.20.2 Constructor & Destructor Documentation

14.20.2.1 QwtCompass() `QwtCompass::QwtCompass ( QWidget * parent = NULL ) [explicit]`

Constructor.

Parameters

parent	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.

Definition at line 158 of file qwt_compass.cpp.

14.20.3 Member Function Documentation

14.20.3.1 drawRose() `void QwtCompass::drawRose ( QPainter * painter, const QPointF & center, double radius,`

```
            double north,
            QPalette::ColorGroup cg ) const  [protected], [virtual]
```

Draw the compass rose

Parameters

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

Definition at line 218 of file qwt_compass.cpp.

14.20.3.2 drawScaleContents()

```
void QwtCompass::drawScaleContents (
    QPainter * painter,
    const QPointF & center,
    double radius ) const  [override], [protected], [virtual]
```

Draw the contents of the scale

Parameters

painter	Painter
center	Center of the content circle
radius	Radius of the content circle

Reimplemented from QwtDial.

Definition at line 189 of file qwt_compass.cpp.

14.20.3.3 keyPressEvent()

```
void QwtCompass::keyPressEvent (
    QKeyEvent * kev )  [override], [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 QwtAbstractSlider.

Definition at line 271 of file qwt_compass.cpp.

14.20.3.4 rose() [1/2] QwtCompassRose * QwtCompass::rose ( )

Returns

rose

See also

setRose()

Definition at line 257 of file qwt_compass.cpp.

14.20.3.5 rose() [2/2] const QwtCompassRose * QwtCompass::rose ( ) const

Returns

rose

See also

setRose()

Definition at line 248 of file qwt_compass.cpp.

14.20.3.6 setRose() void QwtCompass::setRose (QwtCompassRose * *rose*)

Set a rose for the compass

Parameters

rose	Compass rose

Warning

The rose will be deleted, when a different rose is set or in ~QwtCompass

See also

rose()

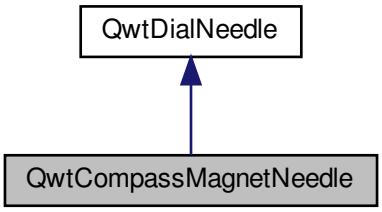
Definition at line 232 of file qwt_compass.cpp.

14.21 QwtCompassMagnetNeedle Class Reference

A magnet needle for compass widgets.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtCompassMagnetNeedle:



Public Types

- enum Style { TriangleStyle , ThinStyle }
  Style of the needle.

Public Member Functions

- QwtCompassMagnetNeedle (Style=TriangleStyle, const QColor &light=Qt::white, const QColor &dark=Qt::red)
  Constructor.

Protected Member Functions

- virtual void drawNeedle (QPainter *, double length, QPalette::ColorGroup) const override

14.21.1 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:

- QPalette::Light
  Used for pointing south
- QPalette::Dark
  Used for pointing north
- QPalette::Base
  Knob (ThinStyle only)

See also

QwtDial, QwtCompass

Definition at line 127 of file qwt_dial_needle.h.

14.21.2 Member Enumeration Documentation

14.21.2.1 Style `enum QwtCompassMagnetNeedle::Style`

Style of the needle.

Enumerator

TriangleStyle	A needle with a triangular shape.
ThinStyle	A thin needle.

Definition at line 131 of file qwt_dial_needle.h.

14.21.3 Member Function Documentation

14.21.3.1 drawNeedle() `void QwtCompassMagnetNeedle::drawNeedle ( QPainter * painter, double length, QPalette::ColorGroup colorGroup ) const [override], [protected], [virtual]`

Draw the needle

Parameters

painter	Painter
length	Length of the needle
colorGroup	Color group, used for painting

Implements QwtDialNeedle.

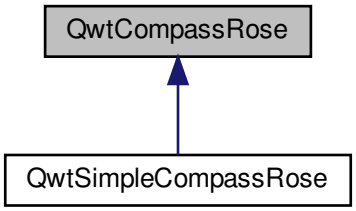
Definition at line 369 of file qwt_dial_needle.cpp.

14.22 QwtCompassRose Class Reference

Abstract base class for a compass rose.

`#include <qwt_compass_rose.h>`

Inheritance diagram for QwtCompassRose:



Public Member Functions

- QwtCompassRose ()

  Constructor.
- virtual ~QwtCompassRose ()

  Destructor.
- virtual void setPalette (const QPalette &)

  Assign a palette.
- const QPalette & palette () const
- virtual void draw (QPainter *painter, const QPointF &center, double radius, double north, QPalette::ColorGroup colorGroup=QPalette::Active) const =0

14.22.1 Detailed Description

Abstract base class for a compass rose.

Definition at line 21 of file qwt_compass_rose.h.

14.22.2 Member Function Documentation

14.22.2.1 draw()

```
virtual void QwtCompassRose::draw (
        QPainter * painter,
        const QPointF & center,
        double radius,
        double north,
        QPalette::ColorGroup colorGroup = QPalette::Active ) const  [pure virtual]
```

Draw the rose

Parameters

painter	Painter
center	Center point
radius	Radius of the rose
north	Position
colorGroup	Color group

Implemented in QwtSimpleCompassRose.

14.22.2.2 palette() `const QPalette & QwtCompassRose::palette ( ) const`

Returns

Current palette

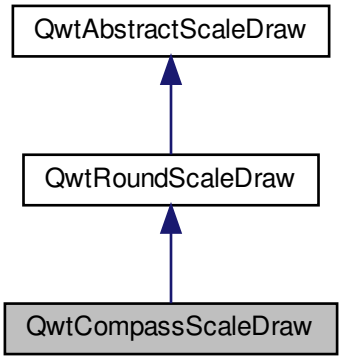
Definition at line 50 of file qwt_compass_rose.cpp.

14.23 QwtCompassScaleDraw Class Reference

A special scale draw made for QwtCompass.

`#include <qwt_compass.h>`

Inheritance diagram for QwtCompassScaleDraw:



Public Member Functions

- QwtCompassScaleDraw ()
  Constructor.
- QwtCompassScaleDraw (const QMap< double, QString > &map)
  Constructor.
- virtual ~QwtCompassScaleDraw ()
  Destructor.
- void setLabelMap (const QMap< double, QString > &map)
  Set a map, mapping values to labels.
- QMap< double, QString > labelMap () const
- virtual QwtText label (double value) const override

Additional Inherited Members

14.23.1 Detailed Description

A special scale draw made for QwtCompass.

QwtCompassScaleDraw maps values to strings using a special map, that can be modified by the application

The default map consists of the labels N, NE, E, SE, S, SW, W, NW.

See also

QwtCompass

Definition at line 32 of file qwt_compass.h.

14.23.2 Constructor & Destructor Documentation

14.23.2.1 QwtCompassScaleDraw() [1/2] QwtCompassScaleDraw::QwtCompassScaleDraw ( ) [explicit]

Constructor.

Initializes a label map for multiples of 45 degrees

Definition at line 28 of file qwt_compass.cpp.

14.23.2.2 QwtCompassScaleDraw() [2/2] QwtCompassScaleDraw::QwtCompassScaleDraw (const QMap< double, QString > & *map*) [explicit]

Constructor.

Parameters

map	Value to label map

Definition at line 63 of file qwt_compass.cpp.

14.23.3 Member Function Documentation

14.23.3.1 label() QwtText QwtCompassScaleDraw::label (double *value*) const [override], [virtual]

Map a value to a corresponding label

Parameters

value	Value that will be mapped

label() looks in the labelMap() for a corresponding label for value or returns an null text.

Returns

Label

See also

labelMap(), setLabelMap()

Reimplemented from QwtAbstractScaleDraw.

Definition at line 116 of file qwt_compass.cpp.

14.23.3.2 labelMap() `QMap< double, QString > QwtCompassScaleDraw::labelMap ( ) const`

Returns

map, mapping values to labels

See also

setLabelMap()

Definition at line 99 of file qwt_compass.cpp.

14.23.3.3 setLabelMap() `void QwtCompassScaleDraw::setLabelMap ( const QMap< double, QString > & map )`

Set a map, mapping values to labels.

Parameters

map	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()

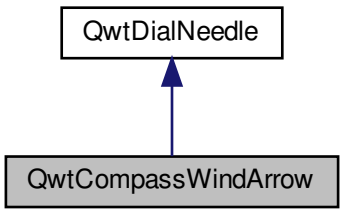
Definition at line 90 of file qwt_compass.cpp.

14.24 QwtCompassWindArrow Class Reference

An indicator for the wind direction.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtCompassWindArrow:



Public Types

- enum Style { Style1 , Style2 }
  Style of the arrow.

Public Member Functions

- QwtCompassWindArrow (Style, const QColor &light=Qt::white, const QColor &dark=Qt::gray)

Protected Member Functions

- virtual void drawNeedle (QPainter *, double length, QPalette::ColorGroup) const override

14.24.1 Detailed Description

An indicator for the wind direction.

QwtCompassWindArrow shows the direction where the wind comes from.

- QPalette::Light
  Used for Style1, or the light half of Style2
- QPalette::Dark
  Used for the dark half of Style2

See also

QwtDial, QwtCompass

Definition at line 164 of file qwt_dial_needle.h.

14.24.2 Member Enumeration Documentation

14.24.2.1 Style `enum QwtCompassWindArrow::Style`

Style of the arrow.

Enumerator

Style1	A needle pointing to the center.
Style2	A needle pointing to the center.

Definition at line 168 of file qwt_dial_needle.h.

14.24.3 Constructor & Destructor Documentation

14.24.3.1 QwtCompassWindArrow()

```
QwtCompassWindArrow::QwtCompassWindArrow (
        Style style,
        const QColor & light = Qt::white,
        const QColor & dark = Qt::gray )
```

Constructor

Parameters

style	Arrow style
light	Light color
dark	Dark color

Definition at line 409 of file qwt_dial_needle.cpp.

14.24.4 Member Function Documentation

14.24.4.1 drawNeedle()

```
void QwtCompassWindArrow::drawNeedle (
        QPainter * painter,
        double length,
        QPalette::ColorGroup colorGroup ) const  [override], [protected], [virtual]
```

Draw the needle

Parameters

painter	Painter
length	Length of the needle
colorGroup	Color group, used for painting

Implements QwtDialNeedle.

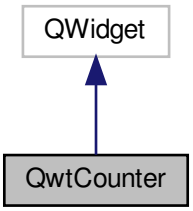
Definition at line 427 of file qwt_dial_needle.cpp.

14.25 QwtCounter Class Reference

The Counter Widget.

```
#include <qwt_counter.h>
```

Inheritance diagram for QwtCounter:



Public Types

- enum Button { Button1 , Button2 , Button3 , ButtonCnt }

  Button index.

Public Slots

- void setValue (double)

  Set a new value without adjusting to the step raster.

Signals

- void buttonReleased (double value)
- void valueChanged (double value)

Public Member Functions

- QwtCounter (QWidget *parent=NULL)
- virtual ~QwtCounter ()

  Destructor.
- void setValid (bool)
- bool isValid () const
- void setWrapping (bool)

  En/Disable wrapping.
- bool wrapping () const
- bool isReadOnly () const
- void setReadOnly (bool)

  Allow/disallow the user to manually edit the value.
- void setNumButtons (int)
- int numButtons () const
- void setIncSteps (QwtCounter::Button, int numSteps)
- int incSteps (QwtCounter::Button) const
- virtual QSize sizeHint () const override

  A size hint.
- double singleStep () const
- void setSingleStep (double stepSize)

  Set the step size of the counter.
- void setRange (double min, double max)

  Set the minimum and maximum values.
- double minimum () const
- void setMinimum (double)
- double maximum () const
- void setMaximum (double)
- void setStepButton1 (int nSteps)
- int stepButton1 () const

  returns the number of increment steps for button 1
- void setStepButton2 (int nSteps)
- int stepButton2 () const

  returns the number of increment steps for button 2
- void setStepButton3 (int nSteps)
- int stepButton3 () const

  returns the number of increment steps for button 3
- double value () const

Protected Member Functions

- virtual bool event (QEvent ∗) override
- virtual void wheelEvent (QWheelEvent ∗) override
- virtual void keyPressEvent (QKeyEvent ∗) override

14.25.1 Detailed Description

The Counter Widget.

A Counter consists of a label displaying a number and one ore 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. When the wrapping property is set the counter is circular.

The number of steps by which a button increments or decrements the value can be specified using setIncSteps(). The number of buttons can be changed with setNumButtons().

Example:

```
#include <qwt_counter.h>
  QwtCounter *counter = new QwtCounter(parent);
  counter->setRange(0.0, 100.0);                    // From 0.0 to 100
  counter->setSingleStep( 1.0 );                    // Step size 1.0
  counter->setNumButtons(2);                        // Two buttons each side
  counter->setIncSteps(QwtCounter::Button1, 1);     // Button 1 increments 1 step
  counter->setIncSteps(QwtCounter::Button2, 20);    // Button 2 increments 20 steps
  connect(counter, SIGNAL(valueChanged(double)), myClass, SLOT(newValue(double)));
```

Definition at line 48 of file qwt_counter.h.

14.25.2 Member Enumeration Documentation

14.25.2.1 Button enum QwtCounter::Button

Button index.

Enumerator

Button1	Button intended for minor steps.
Button2	Button intended for medium steps.
Button3	Button intended for large steps.
ButtonCnt	Number of buttons.

Definition at line 67 of file qwt_counter.h.

14.25.3 Constructor & Destructor Documentation

14.25.3.1 QwtCounter() QwtCounter::QwtCounter (QWidget * *parent* = *NULL*) [explicit]

The counter is initialized with a range is set to [0.0, 1.0] with 0.01 as single step size. The value is invalid.

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

parent	

Definition at line 65 of file qwt_counter.cpp.

14.25.4 Member Function Documentation

14.25.4.1 buttonReleased void QwtCounter::buttonReleased (double *value*) [signal]

This signal is emitted when a button has been released

Parameters

value	The new value

14.25.4.2 event() bool QwtCounter::event (QEvent * *event*) [override], [protected], [virtual]

Handle QEvent::PolishRequest events

Parameters

event	Event

Returns

see QWidget::event()

Definition at line 485 of file qwt_counter.cpp.

14.25.4.3 incSteps() `int QwtCounter::incSteps ( QwtCounter::Button button ) const`

Returns

The number of steps by which a specified button increments the value or 0 if the button is invalid.

Parameters

button	Button index

See also

setIncSteps()

Definition at line 416 of file qwt_counter.cpp.

14.25.4.4 isReadOnly() `bool QwtCounter::isReadOnly ( ) const`

Returns

True, when the line line edit is read only. (default is no)

See also

setReadOnly()

Definition at line 186 of file qwt_counter.cpp.

14.25.4.5 isValid() `bool QwtCounter::isValid ( ) const`

Returns

True, if the value is valid

See also

setValid(), setValue()

Definition at line 166 of file qwt_counter.cpp.

14.25.4.6 keyPressEvent() `void QwtCounter::keyPressEvent ( QKeyEvent * event ) [override], [protected], [virtual]`

Handle key events

- Ctrl + Qt::Key_Home
  Step to minimum()
- Ctrl + Qt::Key_End
  Step to maximum()
- 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)

Parameters

event	Key event

Definition at line 524 of file qwt_counter.cpp.

14.25.4.7 maximum() `double QwtCounter::maximum ( ) const`

Returns

The maximum of the range

See also

setRange(), setMaximum(), minimum()

Definition at line 306 of file qwt_counter.cpp.

14.25.4.8 minimum() `double QwtCounter::minimum ( ) const`

Returns

The minimum of the range

See also

setRange(), setMinimum(), maximum()

Definition at line 286 of file qwt_counter.cpp.

14.25.4.9 numButtons() `int QwtCounter::numButtons ( ) const`

Returns

The number of buttons on each side of the widget.

See also

setNumButtons()

Definition at line 388 of file qwt_counter.cpp.

14.25.4.10 setIncSteps() `void QwtCounter::setIncSteps ( QwtCounter::Button button, int numSteps )`

Specify the number of steps by which the value is incremented or decremented when a specified button is pushed.

Parameters

button	Button index
numSteps	Number of steps

See also

incSteps()

Definition at line 403 of file qwt_counter.cpp.

14.25.4.11 setMaximum() `void QwtCounter::setMaximum ( double value )`

Set the maximum value of the range

Parameters

value	Maximum value

See also

setRange(), setMinimum(), maximum()

Definition at line 297 of file qwt_counter.cpp.

14.25.4.12 setMinimum() `void QwtCounter::setMinimum ( double value )`

Set the minimum value of the range

Parameters

value	Minimum value

See also

setRange(), setMaximum(), minimum()

Note

The maximum is adjusted if necessary to ensure that the range remains valid.

Definition at line 277 of file qwt_counter.cpp.

14.25.4.13 setNumButtons() `void QwtCounter::setNumButtons ( int numButtons )`

Specify the number of buttons on each side of the label

Parameters

numButtons	Number of buttons

See also

numButtons()

Definition at line 362 of file qwt_counter.cpp.

14.25.4.14 setRange() `void QwtCounter::setRange (`

```
double min,
double max )
```

Set the minimum and maximum values.

The maximum is adjusted if necessary to ensure that the range remains valid. The value might be modified to be inside of the range.

Parameters

min	Minimum value
max	Maximum value

See also

minimum(), maximum()

Definition at line 241 of file qwt_counter.cpp.

14.25.4.15 setReadOnly() `void QwtCounter::setReadOnly (`

```
bool on )
```

Allow/disallow the user to manually edit the value.

Parameters

on	True disable editing

See also

isReadOnly()

Definition at line 177 of file qwt_counter.cpp.

14.25.4.16 setSingleStep() `void QwtCounter::setSingleStep (`

```
double stepSize )
```

Set the step size of the counter.

A value <= 0.0 disables stepping

Parameters

stepSize	Single step size

See also

singleStep()

Definition at line 319 of file qwt_counter.cpp.

14.25.4.17 setStepButton1() void QwtCounter::setStepButton1 (int *nSteps*)

Set the number of increment steps for button 1

Parameters

nSteps	Number of steps

Definition at line 429 of file qwt_counter.cpp.

14.25.4.18 setStepButton2() void QwtCounter::setStepButton2 (int *nSteps*)

Set the number of increment steps for button 2

Parameters

nSteps	Number of steps

Definition at line 444 of file qwt_counter.cpp.

14.25.4.19 setStepButton3() void QwtCounter::setStepButton3 (int *nSteps*)

Set the number of increment steps for button 3

Parameters

nSteps	Number of steps

Definition at line 459 of file qwt_counter.cpp.

14.25.4.20 setValid() void QwtCounter::setValid (bool *on*)

Set the counter to be in valid/invalid state

When the counter is set to invalid, no numbers are displayed and the buttons are disabled.

Parameters

on	If true the counter will be set as valid

See also

setValue(), isValid()

Definition at line 142 of file qwt_counter.cpp.

14.25.4.21 setValue `void QwtCounter::setValue ( double value ) [slot]`

Set a new value without adjusting to the step raster.

The state of the counter is set to be valid.

Parameters

value	New value

See also

isValid(), value(), valueChanged()

Warning

The value is clipped when it lies outside the range.

Definition at line 202 of file qwt_counter.cpp.

14.25.4.22 setWrapping() `void QwtCounter::setWrapping ( bool on )`

En/Disable wrapping.

If wrapping is true stepping up from maximum() value will take you to the minimum() value and vice versa.

Parameters

on	En/Disable wrapping

See also

wrapping()

Definition at line 342 of file qwt_counter.cpp.

14.25.4.23 singleStep() `double QwtCounter::singleStep ( ) const`

Returns

Single step size

See also

setSingleStep()

Definition at line 328 of file qwt_counter.cpp.

14.25.4.24 value() `double QwtCounter::value ( ) const`

Returns

Current value of the counter

See also

setValue(), valueChanged()

Definition at line 225 of file qwt_counter.cpp.

14.25.4.25 valueChanged `void QwtCounter::valueChanged ( double value ) [signal]`

This signal is emitted when the counter's value has changed

Parameters

value	The new value

14.25.4.26 wheelEvent() `void QwtCounter::wheelEvent ( QWheelEvent * event ) [override], [protected], [virtual]`

Handle wheel events

Parameters

event	Wheel event

Definition at line 591 of file qwt_counter.cpp.

14.25.4.27 wrapping() `bool QwtCounter::wrapping ( ) const`

Returns

True, when wrapping is set

See also

setWrapping()

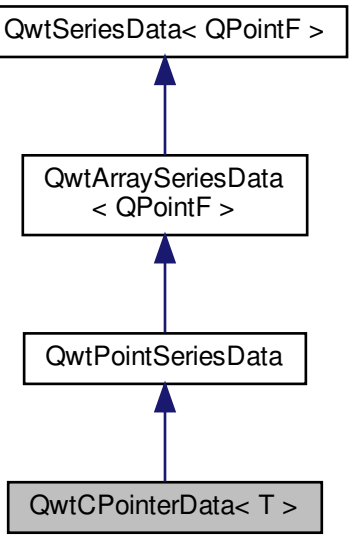
Definition at line 351 of file qwt_counter.cpp.

14.26 QwtCPointerData< T > Class Template Reference

Data class containing two pointers to memory blocks of T.

`#include <qwt_point_data.h>`

Inheritance diagram for QwtCPointerData< T >:



Public Member Functions

- QwtCPointerData (const T ∗x, const T ∗y, size_t size)
- virtual size_t size () const override
- virtual QPointF sample (size_t index) const override
- const T ∗ xData () const
- const T ∗ yData () const

Additional Inherited Members

14.26.1 Detailed Description

template<typename T>
class QwtCPointerData< T >

Data class containing two pointers to memory blocks of T.

Definition at line 43 of file qwt_point_data.h.

14.26.2 Constructor & Destructor Documentation

14.26.2.1 QwtCPointerData() `template<typename T >`

```
QwtCPointerData< T >::QwtCPointerData (
            const T * x,
            const T * y,
            size_t size )
```

Constructor

Parameters

x	Array of x values
y	Array of y values
size	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::setRawSamples()

Definition at line 326 of file qwt_point_data.h.

14.26.3 Member Function Documentation

14.26.3.1 sample() template<typename T >

```
QPointF QwtCPointerData< T >::sample (
            size_t index ) const  [override], [virtual]
```

Return the sample at position i

Parameters

index	Index

Returns

Sample at position i

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 347 of file qwt_point_data.h.

14.26.3.2 size() template<typename T >

```
size_t QwtCPointerData< T >::size  [override], [virtual]
```

Returns

Size of the data set

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 335 of file qwt_point_data.h.

14.26.3.3 xData() template<typename T >

```
const T * QwtCPointerData< T >::xData
```

Returns

Array of the x-values

Definition at line 354 of file qwt_point_data.h.

14.26.3.4 yData() template<typename T >

const T * QwtCPointerData< T >::yData

Returns

Array of the y-values

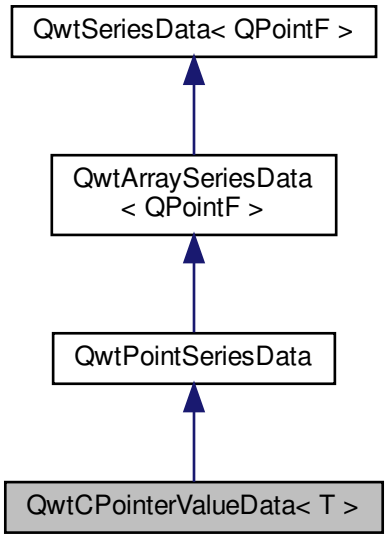
Definition at line 361 of file qwt_point_data.h.

14.27 QwtCPointerValueData< T > Class Template Reference

Data class containing a pointer to memory of y coordinates.

#include <qwt_point_data.h>

Inheritance diagram for QwtCPointerValueData< T >:



Public Member Functions

- QwtCPointerValueData (const T *y, size_t size)
- virtual size_t size () const override
- virtual QPointF sample (size_t index) const override
- const T * yData () const

Additional Inherited Members

14.27.1 Detailed Description

template<typename T>
class QwtCPointerValueData< T >

Data class containing a pointer to memory of y coordinates.

The memory contains the y coordinates, while the index is interpreted as x coordinate.

Definition at line 89 of file qwt_point_data.h.

14.27.2 Constructor & Destructor Documentation

14.27.2.1 QwtCPointerValueData() template<typename T >

```
QwtCPointerValueData< T >::QwtCPointerValueData (
            const T * y,
            size_t size )
```

Constructor

Parameters

y	Array of y values
size	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 QwtCPointerValueData object.

See also

QwtPlotCurve::setData(), QwtPlotCurve::setRawSamples()

Definition at line 380 of file qwt_point_data.h.

14.27.3 Member Function Documentation

14.27.3.1 sample() template<typename T >

```
QPointF QwtCPointerValueData< T >::sample (
            size_t index ) const  [override], [virtual]
```

Return the sample at position i

Parameters

index	Index

Returns

Sample at position i

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 400 of file qwt_point_data.h.

14.27.3.2 size() template<typename T >

```
size_t QwtCPointerValueData< T >::size  [override], [virtual]
```

Returns

Size of the data set

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 388 of file qwt_point_data.h.

14.27.3.3 yData() template<typename T >

```
const T * QwtCPointerValueData< T >::yData
```

Returns

Array of the y-values

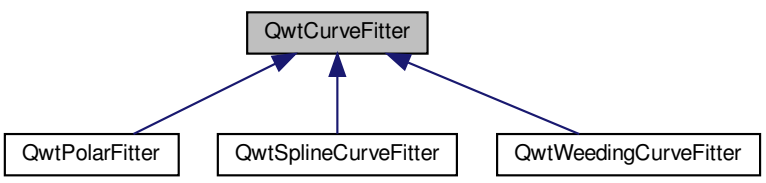
Definition at line 407 of file qwt_point_data.h.

14.28 QwtCurveFitter Class Reference

Abstract base class for a curve fitter.

```
#include <qwt_curve_fitter.h>
```

Inheritance diagram for QwtCurveFitter:



Public Types

- enum Mode { Polygon , Path }

  Preferred mode of the fitting algorithm.

Public Member Functions

- virtual ~QwtCurveFitter ()

  Destructor.
- Mode mode () const
- virtual QPolygonF fitCurve (const QPolygonF &polygon) const =0
- virtual QPainterPath fitCurvePath (const QPolygonF &polygon) const =0

Protected Member Functions

- QwtCurveFitter (Mode mode)

14.28.1 Detailed Description

Abstract base class for a curve fitter.

Definition at line 21 of file qwt_curve_fitter.h.

14.28.2 Member Enumeration Documentation

14.28.2.1 Mode `enum QwtCurveFitter::Mode`

Preferred mode of the fitting algorithm.

Even if a QPainterPath can always be created from a QPolygonF the overhead of the conversion can be avoided by indicating the preference of the implementation to the application code.

Enumerator

Polygon	The fitting algorithm creates a polygon - the implementation of fitCurvePath() simply wraps the polygon into a path. **See also** QwtWeedingCurveFitter
Path	The fitting algorithm creates a painter path - the implementation of fitCurve() extracts a polygon from the path. **See also** QwtSplineCurveFitter

Definition at line 32 of file qwt_curve_fitter.h.

14.28.3 Constructor & Destructor Documentation

14.28.3.1 QwtCurveFitter() `QwtCurveFitter::QwtCurveFitter ( Mode mode ) [explicit], [protected]`

Constructor

Parameters

mode	Preferred fitting mode

Definition at line 16 of file qwt_curve_fitter.cpp.

14.28.4 Member Function Documentation

14.28.4.1 fitCurve() `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

polygon	Series of data points

Returns

Curve points

See also

fitCurvePath()

Implemented in QwtWeedingCurveFitter, QwtSplineCurveFitter, and QwtPolarFitter.

14.28.4.2 fitCurvePath() `virtual QPainterPath QwtCurveFitter::fitCurvePath ( const QPolygonF & polygon ) const [pure virtual]`

Find a curve path which has the best fit to a series of data points

Parameters

polygon	Series of data points

Returns

Curve path

See also

fitCurve()

Implemented in QwtWeedingCurveFitter, QwtSplineCurveFitter, and QwtPolarFitter.

14.28.4.3 mode() `QwtCurveFitter::Mode QwtCurveFitter::mode ( ) const`

Returns

Preferred fitting mode

Definition at line 27 of file qwt_curve_fitter.cpp.

14.29 QwtDate Class Reference

A collection of methods around date/time values.

`#include <qwt_date.h>`

Public Types

- enum Week0Type { FirstThursday , FirstDay }
- enum IntervalType {
  Millisecond , Second , Minute , Hour ,
  Day , Week , Month , Year }
- enum { JulianDayForEpoch = 2440588 }

Static Public Member Functions

- static QDate minDate ()
- static QDate maxDate ()
- static QDateTime toDateTime (double value, Qt::TimeSpec=Qt::UTC)
- static double toDouble (const QDateTime &)
- static QDateTime ceil (const QDateTime &, IntervalType)
- static QDateTime floor (const QDateTime &, IntervalType)
- static QDate dateOfWeek0 (int year, Week0Type)

  Date of the first day of the first week for a year.
- static int weekNumber (const QDate &, Week0Type)
- static int utcOffset (const QDateTime &)
- static QString toString (const QDateTime &, const QString &format, Week0Type)

14.29.1 Detailed Description

A collection of methods around date/time values.

Qt offers convenient classes for dealing with date/time values, but Qwt uses coordinate systems that are based on doubles. QwtDate offers methods to translate from QDateTime to double and v.v.

A double is interpreted as the number of milliseconds since 1970-01-01T00:00:00 Universal Coordinated Time - also known as "The Epoch".

While the range of the Julian day in Qt4 is limited to [0, MAX_INT], Qt5 stores it as qint64 offering a huge range of valid dates. As the significance of a double is below this (assuming a fraction of 52 bits) the translation is not bijective with rounding errors for dates very far from Epoch. For a resolution of 1 ms those start to happen for dates above the year 144683.

An axis for a date/time interval is expected to be aligned and divided in time/date units like seconds, minutes, ... QwtDate offers several algorithms that are needed to calculate these axes.

See also

QwtDateScaleEngine, QwtDateScaleDraw, QDate, QTime

Definition at line 42 of file qwt_date.h.

14.29.2 Member Enumeration Documentation

14.29.2.1 anonymous enum `anonymous enum`

Enumerator

JulianDayForEpoch	The Julian day of "The Epoch".

Definition at line 102 of file qwt_date.h.

14.29.2.2 IntervalType `enum QwtDate::IntervalType`

Classification of an time interval

Time intervals needs to be classified to decide how to align and divide it.

Enumerator

Millisecond	The interval is related to milliseconds.
Second	The interval is related to seconds.
Minute	The interval is related to minutes.
Hour	The interval is related to hours.

Enumerator

Day	The interval is related to days.
Week	The interval is related to weeks.
Month	The interval is related to months.
Year	The interval is related to years.

Definition at line 75 of file qwt_date.h.

14.29.2.3 Week0Type `enum QwtDate::Week0Type`

How to identify the first week of year differs between countries.

Enumerator

FirstThursday	According to ISO 8601 the first week of a year is defined as "the week with the year's first Thursday in it". FirstThursday corresponds to the numbering that is implemented in QDate::weekNumber().
FirstDay	"The week with January 1.1 in it." In the U.S. this definition is more common than FirstThursday.

Definition at line 49 of file qwt_date.h.

14.29.3 Member Function Documentation

14.29.3.1 ceil() `QDateTime QwtDate::ceil ( const QDateTime & dateTime, IntervalType intervalType ) [static]`

Ceil a datetime according the interval type

Parameters

dateTime	Datetime value
intervalType	Interval type, how to ceil. F.e. when intervalType = QwtDate::Months, the result will be ceiled to the next beginning of a month

Returns

Ceiled datetime

See also

floor()

Definition at line 323 of file qwt_date.cpp.

14.29.3.2 dateOfWeek0()

```
QDate QwtDate::dateOfWeek0 (
    int year,
    Week0Type type ) [static]
```

Date of the first day of the first week for a year.

The first day of a week depends on the current locale (QLocale::firstDayOfWeek()).

Parameters

year	Year
type	Option how to identify the first week

Returns

First day of week 0

See also

QLocale::firstDayOfWeek(), weekNumber()

Definition at line 542 of file qwt_date.cpp.

14.29.3.3 floor()

```
QDateTime QwtDate::floor (
    const QDateTime & dateTime,
    IntervalType intervalType ) [static]
```

Floor a datetime according the interval type

Parameters

dateTime	Datetime value
intervalType	Interval type, how to ceil. F.e. when intervalType = QwtDate::Months, the result will be ceiled to the next beginning of a month

Returns

Floored datetime

See also

floor()

Definition at line 425 of file qwt_date.cpp.

14.29.3.4 maxDate() `QDate QwtDate::maxDate ( ) [static]`

Maximum for the supported date range

The range of valid dates depends on how QDate stores the Julian day internally.

- For Qt4 it is "Tue Jun 3 5874898"
- For Qt5 it is "Tue Dec 31 2147483647"

Returns

maximum of the date range

See also

minDate()

Note

The maximum differs between Qt4 and Qt5

Definition at line 521 of file qwt_date.cpp.

14.29.3.5 minDate() `QDate QwtDate::minDate ( ) [static]`

Minimum for the supported date range

The range of valid dates depends on how QDate stores the Julian day internally.

- For Qt4 it is "Tue Jan 2 -4713"
- For Qt5 it is "Thu Jan 1 -2147483648"

Returns

minimum of the date range

See also

maxDate()

Definition at line 499 of file qwt_date.cpp.

14.29.3.6 toDateTime() `QDateTime QwtDate::toDateTime ( double value, Qt::TimeSpec timeSpec = Qt::UTC ) [static]`

Translate from double to QDateTime

Parameters

value	Number of milliseconds since the epoch, 1970-01-01T00:00:00 UTC
timeSpec	Time specification

Returns

Datetime value

See also

toDouble(), QDateTime::setMSecsSinceEpoch()

Note

The return datetime for Qt::OffsetFromUTC will be Qt::UTC

Definition at line 261 of file qwt_date.cpp.

14.29.3.7 toDouble() `double QwtDate::toDouble ( const QDateTime & dateTime ) [static]`

Translate from QDateTime to double

Parameters

dateTime	Datetime value

Returns

Number of milliseconds since 1970-01-01T00:00:00 UTC has passed.

See also

toDateTime(), QDateTime::toMSecsSinceEpoch()

Warning

For values very far below or above 1970-01-01 UTC rounding errors will happen due to the limited significance of a double.

Definition at line 298 of file qwt_date.cpp.

14.29.3.8 toString()

```
QString QwtDate::toString (
    const QDateTime & dateTime,
    const QString & format,
    Week0Type week0Type ) [static]
```

Translate a datetime into a string

Beside the format expressions documented in QDateTime::toString() the following expressions are supported:

- w
  week number: (1 - 53)
- ww
  week number with a leading zero (01 - 53)

As week 1 usually starts in the previous year a special rule is applied for formats, where the year is expected to match the week number - even if the date belongs to the previous year.

Parameters

dateTime	Datetime value
format	Format string
week0Type	Specification of week 0

Returns

Datetime string

See also

QDateTime::toString(), weekNumber(), QwtDateScaleDraw

Definition at line 686 of file qwt_date.cpp.

14.29.3.9 utcOffset()

```
int QwtDate::utcOffset (
    const QDateTime & dateTime ) [static]
```

Offset in seconds from Coordinated Universal Time

The offset depends on the time specification of dateTime:

- Qt::UTC 0, dateTime has no offset
- Qt::OffsetFromUTC returns dateTime.offsetFromUtc()
- Qt::LocalTime: number of seconds from the UTC

For Qt::LocalTime the offset depends on the timezone and daylight savings.

Parameters

dateTime	Datetime value

Returns

Offset in seconds

Definition at line 635 of file qwt_date.cpp.

14.29.3.10 weekNumber() `int QwtDate::weekNumber ( const QDate & date, Week0Type type ) [static]`

Find the week number of a date

- QwtDate::FirstThursday
  Corresponding to ISO 8601 (see QDate::weekNumber()).
- QwtDate::FirstDay
  Number of weeks that have begun since dateOfWeek0().

Parameters

date	Date
type	Option how to identify the first week

Returns

Week number, starting with 1

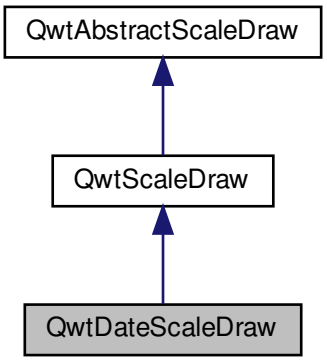
Definition at line 585 of file qwt_date.cpp.

14.30 QwtDateScaleDraw Class Reference

A class for drawing datetime scales.

`#include <qwt_date_scale_draw.h>`

Inheritance diagram for QwtDateScaleDraw:



Public Member Functions

- QwtDateScaleDraw (Qt::TimeSpec=Qt::LocalTime)

  Constructor.
- virtual ~QwtDateScaleDraw ()

  Destructor.
- void setDateFormat (QwtDate::IntervalType, const QString &)
- QString dateFormat (QwtDate::IntervalType) const
- void setTimeSpec (Qt::TimeSpec)
- Qt::TimeSpec timeSpec () const
- void setUtcOffset (int seconds)
- int utcOffset () const
- void setWeek0Type (QwtDate::Week0Type)
- QwtDate::Week0Type week0Type () const
- virtual QwtText label (double) const override

  Convert a value into its representing label.
- QDateTime toDateTime (double) const

Protected Member Functions

- virtual QwtDate::IntervalType intervalType (const QwtScaleDiv &) const
- virtual QString dateFormatOfDate (const QDateTime &, QwtDate::IntervalType) const

Additional Inherited Members

14.30.1 Detailed Description

A class for drawing datetime scales.

QwtDateScaleDraw displays values as datetime labels. The format of the labels depends on the alignment of the major tick labels.

The default format strings are:

- Millisecond
  "hh:mm:ss:zzz\nddd dd MMM yyyy"
- Second
  "hh:mm:ss\nddd dd MMM yyyy"
- Minute
  "hh:mm\nddd dd MMM yyyy"
- Hour
  "hh:mm\nddd dd MMM yyyy"
- Day
  "ddd dd MMM yyyy"
- Week
  "Www yyyy"
- Month
  "MMM yyyy"
- Year
  "yyyy"

The format strings can be modified using setDateFormat() or individually for each tick label by overloading dateFormatOfDate(),

Usually QwtDateScaleDraw is used in combination with QwtDateScaleEngine, that calculates scales for datetime intervals.

See also

QwtDateScaleEngine, QwtPlot::setAxisScaleDraw()

Definition at line 52 of file qwt_date_scale_draw.h.

14.30.2 Constructor & Destructor Documentation

14.30.2.1 QwtDateScaleDraw() `QwtDateScaleDraw::QwtDateScaleDraw ( Qt::TimeSpec timeSpec = Qt::LocalTime ) [explicit]`

Constructor.

The default setting is to display tick labels for the given time specification. The first week of a year is defined like for QwtDate::FirstThursday.

Parameters

timeSpec	Time specification

See also

setTimeSpec(), setWeek0Type()

Definition at line 48 of file qwt_date_scale_draw.cpp.

14.30.3 Member Function Documentation

14.30.3.1 dateFormat() `QString QwtDateScaleDraw::dateFormat ( QwtDate::IntervalType intervalType ) const`

Parameters

intervalType	Interval type

Returns

Default format string for an datetime interval type

See also

setDateFormat(), dateFormatOfDate()

Definition at line 152 of file qwt_date_scale_draw.cpp.

14.30.3.2 dateFormatOfDate() `QString QwtDateScaleDraw::dateFormatOfDate ( const QDateTime & dateTime, QwtDate::IntervalType intervalType ) const [protected], [virtual]`

Format string for the representation of a datetime

dateFormatOfDate() is intended to be overloaded for situations, where formats are individual for specific datetime values.

The default setting ignores dateTime and return the default format for the interval type.

Parameters

dateTime	Datetime value
intervalType	Interval type

Returns

Format string

See also

setDateFormat(), QwtDate::toString()

Definition at line 180 of file qwt_date_scale_draw.cpp.

14.30.3.3 intervalType()

```
QwtDate::IntervalType QwtDateScaleDraw::intervalType (
    const QwtScaleDiv & scaleDiv ) const  [protected], [virtual]
```

Find the less detailed datetime unit, where no rounding errors happen.

Parameters

scaleDiv	Scale division

Returns

Interval type

See also

dateFormatOfDate()

Definition at line 223 of file qwt_date_scale_draw.cpp.

14.30.3.4 label()

```
QwtText QwtDateScaleDraw::label (
    double value ) const  [override], [virtual]
```

Convert a value into its representing label.

The value is converted to a datetime value using toDateTime() and converted to a plain text using QwtDate::toString().

Parameters

value	Value

Returns

Label string.

See also

dateFormatOfDate()

Reimplemented from QwtAbstractScaleDraw.

Definition at line 205 of file qwt_date_scale_draw.cpp.

14.30.3.5 setDateFormat() `void QwtDateScaleDraw::setDateFormat ( QwtDate::IntervalType intervalType, const QString & format )`

Set the default format string for an datetime interval type

Parameters

intervalType	Interval type
format	Default format string

See also

dateFormat(), dateFormatOfDate(), QwtDate::toString()

Definition at line 137 of file qwt_date_scale_draw.cpp.

14.30.3.6 setTimeSpec() `void QwtDateScaleDraw::setTimeSpec ( Qt::TimeSpec timeSpec )`

Set the time specification used for the tick labels

Parameters

timeSpec	Time specification

See also

timeSpec(), setUtcOffset(), toDateTime()

Definition at line 65 of file qwt_date_scale_draw.cpp.

14.30.3.7 setUtcOffset() `void QwtDateScaleDraw::setUtcOffset ( int seconds )`

Set the offset in seconds from Coordinated Universal Time

Parameters

seconds	Offset in seconds

Note

The offset has no effect beside for the time specification Qt::OffsetFromUTC.

See also

QDate::utcOffset(), setTimeSpec(), toDateTime()

Definition at line 89 of file qwt_date_scale_draw.cpp.

14.30.3.8 setWeek0Type() `void QwtDateScaleDraw::setWeek0Type ( QwtDate::Week0Type week0Type )`

Sets how to identify the first week of a year.

Parameters

week0Type	Mode how to identify the first week of a year

See also

week0Type().

Note

week0Type has no effect beside for intervals classified as QwtDate::Week.

Definition at line 115 of file qwt_date_scale_draw.cpp.

14.30.3.9 timeSpec() `Qt::TimeSpec QwtDateScaleDraw::timeSpec ( ) const`

Returns

Time specification used for the tick labels

See also

setTimeSpec(), utcOffset(), toDateTime()

Definition at line 74 of file qwt_date_scale_draw.cpp.

14.30.3.10 toDateTime() `QDateTime QwtDateScaleDraw::toDateTime ( double value ) const`

Translate a double value into a QDateTime object.

Returns

QDateTime object initialized with timeSpec() and utcOffset().

See also

timeSpec(), utcOffset(), QwtDate::toDateTime()

Definition at line 269 of file qwt_date_scale_draw.cpp.

14.30.3.11 utcOffset() `int QwtDateScaleDraw::utcOffset ( ) const`

Returns

Offset in seconds from Coordinated Universal Time

Note

The offset has no effect beside for the time specification Qt::OffsetFromUTC.

See also

QDate::setUtcOffset(), setTimeSpec(), toDateTime()

Definition at line 101 of file qwt_date_scale_draw.cpp.

14.30.3.12 week0Type() `QwtDate::Week0Type QwtDateScaleDraw::week0Type ( ) const`

Returns

Setting how to identify the first week of a year.

See also

setWeek0Type()

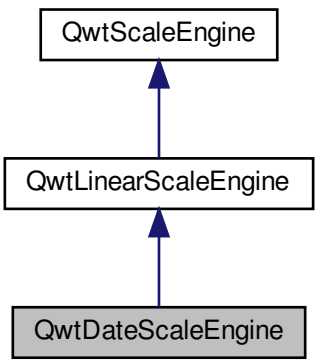
Definition at line 124 of file qwt_date_scale_draw.cpp.

14.31 QwtDateScaleEngine Class Reference

A scale engine for date/time values.

`#include <qwt_date_scale_engine.h>`

Inheritance diagram for QwtDateScaleEngine:



Public Member Functions

- QwtDateScaleEngine (Qt::TimeSpec=Qt::LocalTime)

  Constructor.
- virtual ~QwtDateScaleEngine ()

  Destructor.
- void setTimeSpec (Qt::TimeSpec)
- Qt::TimeSpec timeSpec () const
- void setUtcOffset (int seconds)
- int utcOffset () const
- void setWeek0Type (QwtDate::Week0Type)
- QwtDate::Week0Type week0Type () const
- void setMaxWeeks (int)
- int maxWeeks () const
- virtual void autoScale (int maxNumSteps, double &x1, double &x2, double &stepSize) const override
- virtual QwtScaleDiv divideScale (double x1, double x2, int maxMajorSteps, int maxMinorSteps, double stepSize=0.0) const override

  Calculate a scale division for a date/time interval.
- virtual QwtDate::IntervalType intervalType (const QDateTime &, const QDateTime &, int maxSteps) const
- QDateTime toDateTime (double) const

Protected Member Functions

- virtual QDateTime alignDate (const QDateTime &, double stepSize, QwtDate::IntervalType, bool up) const

Additional Inherited Members

14.31.1 Detailed Description

A scale engine for date/time values.

QwtDateScaleEngine builds scales from a time intervals. Together with QwtDateScaleDraw it can be used for axes according to date/time values.

Years, months, weeks, days, hours and minutes are organized in steps with non constant intervals. QwtDateScaleEngine classifies intervals and aligns the boundaries and tick positions according to this classification.

QwtDateScaleEngine supports representations depending on Qt::TimeSpec specifications. The valid range for scales is limited by the range of QDateTime, that differs between Qt4 and Qt5.

Datetime values are expected as the number of milliseconds since 1970-01-01T00:00:00 Universal Coordinated Time - also known as "The Epoch", that can be converted to QDateTime using QwtDate::toDateTime().

See also

QwtDate, QwtPlot::setAxisScaleEngine(), QwtAbstractScale::setScaleEngine()

Definition at line 42 of file qwt_date_scale_engine.h.

14.31.2 Constructor & Destructor Documentation

14.31.2.1 QwtDateScaleEngine() `QwtDateScaleEngine::QwtDateScaleEngine ( Qt::TimeSpec timeSpec = Qt::LocalTime ) [explicit]`

Constructor.

The engine is initialized to build scales for the given time specification. It classifies intervals $>$ 4 weeks as $>=$ Qt::Month. The first week of a year is defined like for QwtDate::FirstThursday.

Parameters

timeSpec	Time specification

See also

setTimeSpec(), setMaxWeeks(), setWeek0Type()

Definition at line 747 of file qwt_date_scale_engine.cpp.

14.31.3 Member Function Documentation

14.31.3.1 alignDate()

```
QDateTime QwtDateScaleEngine::alignDate (
    const QDateTime & dateTime,
    double stepSize,
    QwtDate::IntervalType intervalType,
    bool up ) const  [protected], [virtual]
```

Align a date/time value for a step size

For Qt::Day alignments there is no "natural day 0" - instead the first day of the year is used to avoid jumping major ticks positions when panning a scale. For other alignments (f.e according to the first day of the month) alignDate() has to be overloaded.

Parameters

dateTime	Date/time value
stepSize	Step size
intervalType	Interval type
up	When true dateTime is ceiled - otherwise it is floored

Returns

Aligned date/time value

Definition at line 1108 of file qwt_date_scale_engine.cpp.

14.31.3.2 autoScale()

```
void QwtDateScaleEngine::autoScale (
    int maxNumSteps,
    double & x1,
    double & x2,
    double & stepSize ) const  [override], [virtual]
```

Align and divide an interval

The algorithm aligns and divides the interval into steps.

Datetime interval divisions are usually not equidistant and the calculated stepSize can only be used as an approximation for the steps calculated by divideScale().

Parameters

maxNumSteps	Max. number of steps
x1	First limit of the interval (In/Out)
x2	Second limit of the interval (In/Out)
stepSize	Step size (Out)

See also

QwtScaleEngine::setAttribute()

Reimplemented from QwtLinearScaleEngine.

Definition at line 925 of file qwt_date_scale_engine.cpp.

14.31.3.3 divideScale()

```
QwtScaleDiv QwtDateScaleEngine::divideScale (
        double x1,
        double x2,
        int maxMajorSteps,
        int maxMinorSteps,
        double stepSize = 0.0 ) const  [override], [virtual]
```

Calculate a scale division for a date/time interval.

Parameters

x1	First interval limit
x2	Second interval limit
maxMajorSteps	Maximum for the number of major steps
maxMinorSteps	Maximum number of minor steps
stepSize	Step size. If stepSize == 0, the scaleEngine calculates one.

Returns

Calculated scale division

Reimplemented from QwtLinearScaleEngine.

Definition at line 992 of file qwt_date_scale_engine.cpp.

14.31.3.4 intervalType()

```
QwtDate::IntervalType QwtDateScaleEngine::intervalType (
        const QDateTime & minDate,
        const QDateTime & maxDate,
        int maxSteps ) const  [virtual]
```

Classification of a date/time interval division

Parameters

minDate	Minimum (= earlier) of the interval
maxDate	Maximum (= later) of the interval
maxSteps	Maximum for the number of steps

Returns

Interval classification

Definition at line 865 of file qwt_date_scale_engine.cpp.

14.31.3.5 maxWeeks() `int QwtDateScaleEngine::maxWeeks ( ) const`

Returns

Upper limit for the number of weeks, when an interval can be classified as Qt::Week.

See also

setMaxWeeks(), week0Type()

Definition at line 851 of file qwt_date_scale_engine.cpp.

14.31.3.6 setMaxWeeks() `void QwtDateScaleEngine::setMaxWeeks ( int weeks )`

Set a upper limit for the number of weeks, when an interval can be classified as Qt::Week.

The default setting is 4 weeks.

Parameters

weeks	Upper limit for the number of weeks

Note

In business charts a year is often divided into weeks [1-52]

See also

maxWeeks(), setWeek0Type()

Definition at line 841 of file qwt_date_scale_engine.cpp.

14.31.3.7 setTimeSpec() `void QwtDateScaleEngine::setTimeSpec ( Qt::TimeSpec timeSpec )`

Set the time specification used by the engine

Parameters

timeSpec	Time specification

See also

timeSpec(), setUtcOffset(), toDateTime()

Definition at line 765 of file qwt_date_scale_engine.cpp.

14.31.3.8 setUtcOffset() `void QwtDateScaleEngine::setUtcOffset ( int seconds )`

Set the offset in seconds from Coordinated Universal Time

Parameters

seconds	Offset in seconds

Note

The offset has no effect beside for the time specification Qt::OffsetFromUTC.

See also

QDate::utcOffset(), setTimeSpec(), toDateTime()

Definition at line 789 of file qwt_date_scale_engine.cpp.

14.31.3.9 setWeek0Type() `void QwtDateScaleEngine::setWeek0Type ( QwtDate::Week0Type week0Type )`

Sets how to identify the first week of a year.

Parameters

week0Type	Mode how to identify the first week of a year

See also

week0Type(), setMaxWeeks()

Note

week0Type has no effect beside for intervals classified as QwtDate::Week.

Definition at line 815 of file qwt_date_scale_engine.cpp.

14.31.3.10 timeSpec() `Qt::TimeSpec QwtDateScaleEngine::timeSpec ( ) const`

Returns

Time specification used by the engine

See also

setTimeSpec(), utcOffset(), toDateTime()

Definition at line 774 of file qwt_date_scale_engine.cpp.

14.31.3.11 toDateTime() `QDateTime QwtDateScaleEngine::toDateTime ( double value ) const`

Translate a double value into a QDateTime object.

For QDateTime result is bounded by QwtDate::minDate() and QwtDate::maxDate()

Returns

QDateTime object initialized with timeSpec() and utcOffset().

See also

timeSpec(), utcOffset(), QwtDate::toDateTime()

Definition at line 1298 of file qwt_date_scale_engine.cpp.

14.31.3.12 utcOffset() `int QwtDateScaleEngine::utcOffset ( ) const`

Returns

Offset in seconds from Coordinated Universal Time

Note

The offset has no effect beside for the time specification Qt::OffsetFromUTC.

See also

QDate::setUtcOffset(), setTimeSpec(), toDateTime()

Definition at line 801 of file qwt_date_scale_engine.cpp.

14.31.3.13 week0Type() `QwtDate::Week0Type QwtDateScaleEngine::week0Type ( ) const`

Returns

Setting how to identify the first week of a year.

See also

setWeek0Type(), maxWeeks()

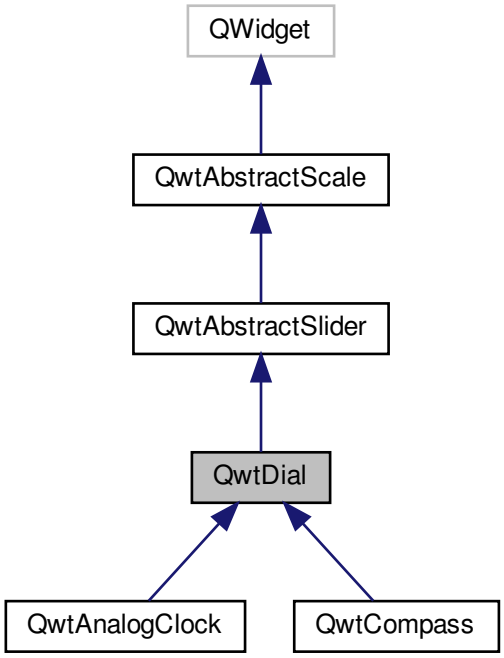
Definition at line 824 of file qwt_date_scale_engine.cpp.

14.32 QwtDial Class Reference

QwtDial class provides a rounded range control.

```
#include <qwt_dial.h>
```

Inheritance diagram for QwtDial:



Public Types

- enum Shadow { Plain = QFrame::Plain , Raised = QFrame::Raised , Sunken = QFrame::Sunken }

  Frame shadow.
- enum Mode { RotateNeedle , RotateScale }

  Mode controlling whether the needle or the scale is rotating.

Public Member Functions

- QwtDial (QWidget *parent=NULL)

  Constructor.
- virtual ~QwtDial ()

  Destructor.
- void setFrameShadow (Shadow)
- Shadow frameShadow () const
- void setLineWidth (int)
- int lineWidth () const
- void setMode (Mode)

  Change the mode of the dial.
- Mode mode () const
- void setScaleArc (double minArc, double maxArc)
- void setMinScaleArc (double)
- double minScaleArc () const

- void setMaxScaleArc (double)
- double maxScaleArc () const
- virtual void setOrigin (double)

  Change the origin.
- double origin () const
- void setNeedle (QwtDialNeedle ∗)
- const QwtDialNeedle ∗ needle () const
- QwtDialNeedle ∗ needle ()
- QRect boundingRect () const
- QRect innerRect () const
- virtual QRect scaleInnerRect () const
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override
- void setScaleDraw (QwtRoundScaleDraw ∗)
- QwtRoundScaleDraw ∗ scaleDraw ()
- const QwtRoundScaleDraw ∗ scaleDraw () const

Protected Member Functions

- virtual void wheelEvent (QWheelEvent ∗) override
- virtual void paintEvent (QPaintEvent ∗) override
- virtual void changeEvent (QEvent ∗) override
- virtual void drawFrame (QPainter ∗)
- virtual void drawContents (QPainter ∗) const

  Draw the contents inside the frame.
- virtual void drawFocusIndicator (QPainter ∗) const
- void invalidateCache ()
- virtual void drawScale (QPainter ∗, const QPointF &center, double radius) const
- virtual void drawScaleContents (QPainter ∗painter, const QPointF &center, double radius) const
- virtual void drawNeedle (QPainter ∗, const QPointF &, double radius, double direction, QPalette::ColorGroup) const
- virtual double scrolledTo (const QPoint &) const override

  Determine the value for a new position of the slider handle.
- virtual bool isScrollPosition (const QPoint &) const override

  Determine what to do when the user presses a mouse button.
- virtual void sliderChange () override

  Calling update()
- virtual void scaleChange () override

Additional Inherited Members

14.32.1 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 QwtAbstractSlider::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.

Often dials have to be updated very often according to values from external devices. For these high refresh rates QwtDial caches as much as possible. For derived classes it might be necessary to clear these caches manually according to attribute changes using invalidateCache().

See also

QwtCompass, QwtAnalogClock, QwtDialNeedle

Note

The controls and dials examples shows different types of dials.

QDial is more similar to QwtKnob than to QwtDial

Definition at line 50 of file qwt_dial.h.

14.32.2 Member Enumeration Documentation

14.32.2.1 Mode `enum QwtDial::Mode`

Mode controlling whether the needle or the scale is rotating.

Enumerator

RotateNeedle	The needle is rotating.
RotateScale	The needle is fixed, the scales are rotating.

Definition at line 86 of file qwt_dial.h.

14.32.2.2 Shadow `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.

Enumerator

Plain	QFrame::Plain.
Raised	QFrame::Raised.
Sunken	QFrame::Sunken.

Definition at line 73 of file qwt_dial.h.

14.32.3 Constructor & Destructor Documentation

14.32.3.1 QwtDial() `QwtDial::QwtDial (`

`QWidget * parent = NULL ) [explicit]`

Constructor.

Parameters

parent	Parent widget

Create a dial widget with no needle. The scale is initialized to [0.0, 360.0] and 360 steps (QwtAbstractSlider::setTotalSteps()). The origin of the scale is at 90°,

The value is set to 0.0.

The default mode is QwtDial::RotateNeedle.

Definition at line 124 of file qwt_dial.cpp.

14.32.4 Member Function Documentation

14.32.4.1 boundingRect() `QRect QwtDial::boundingRect ( ) const`

Returns

bounding rectangle of the dial including the frame

See also

setLineWidth(), scaleInnerRect(), innerRect()

Definition at line 234 of file qwt_dial.cpp.

14.32.4.2 changeEvent() `void QwtDial::changeEvent (`

`QEvent * event ) [override], [protected], [virtual]`

Change Event handler

Parameters

event	Change event

Invalidates internal paint caches if necessary

Reimplemented from QwtAbstractScale.

Definition at line 804 of file qwt_dial.cpp.

14.32.4.3 drawContents() `void QwtDial::drawContents ( QPainter * painter ) const [protected], [virtual]`

Draw the contents inside the frame.

QPalette::Window is the background color outside of the frame. QPalette::Base is the background color inside the frame. QPalette::WindowText is the background color inside the scale.

Parameters

painter	Painter

See also

boundingRect(), innerRect(), scaleInnerRect(), QWidget::setPalette()

Definition at line 391 of file qwt_dial.cpp.

14.32.4.4 drawFocusIndicator() `void QwtDial::drawFocusIndicator ( QPainter * painter ) const [protected], [virtual]`

Draw the focus indicator

Parameters

painter	Painter

Definition at line 362 of file qwt_dial.cpp.

14.32.4.5 drawFrame() `void QwtDial::drawFrame ( QPainter * painter ) [protected], [virtual]`

Draw the frame around the dial

Parameters

painter	Painter

See also

lineWidth(), frameShadow()

Definition at line 373 of file qwt_dial.cpp.

14.32.4.6 drawNeedle()

```
void QwtDial::drawNeedle (
    QPainter * painter,
    const QPointF & center,
    double radius,
    double direction,
    QPalette::ColorGroup colorGroup ) const  [protected], [virtual]
```

Draw the needle

Parameters

painter	Painter
center	Center of the dial
radius	Length for the needle
direction	Direction of the needle in degrees, counter clockwise
colorGroup	ColorGroup

Reimplemented in QwtAnalogClock.

Definition at line 438 of file qwt_dial.cpp.

14.32.4.7 drawScale()

```
void QwtDial::drawScale (
    QPainter * painter,
    const QPointF & center,
    double radius ) const  [protected], [virtual]
```

Draw the scale

Parameters

painter	Painter
center	Center of the dial
radius	Radius of the scale

Definition at line 475 of file qwt_dial.cpp.

14.32.4.8 drawScaleContents()

```
void QwtDial::drawScaleContents (
    QPainter * painter,
    const QPointF & center,
    double radius ) const  [protected], [virtual]
```

Draw the contents inside the scale

Paints nothing.

Parameters

painter	Painter
center	Center of the contents circle
radius	Radius of the contents circle

Reimplemented in QwtCompass.

Definition at line 506 of file qwt_dial.cpp.

14.32.4.9 frameShadow() `QwtDial::Shadow QwtDial::frameShadow ( ) const`

Returns

Frame shadow /sa setFrameShadow(), lineWidth(), QFrame::frameShadow()

Definition at line 186 of file qwt_dial.cpp.

14.32.4.10 innerRect() `QRect QwtDial::innerRect ( ) const`

Returns

bounding rectangle of the circle inside the frame

See also

setLineWidth(), scaleInnerRect(), boundingRect()

Definition at line 224 of file qwt_dial.cpp.

14.32.4.11 invalidateCache() `void QwtDial::invalidateCache ( ) [protected]`

Invalidate the internal caches used to speed up repainting

Definition at line 301 of file qwt_dial.cpp.

14.32.4.12 isScrollPosition() `bool QwtDial::isScrollPosition ( const QPoint & pos ) const [override], [protected], [virtual]`

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

Parameters

pos	Mouse position

Return values

True,when	the inner circle contains pos

See also

scrolledTo()

Implements QwtAbstractSlider.

Definition at line 721 of file qwt_dial.cpp.

14.32.4.13 lineWidth() `int QwtDial::lineWidth ( ) const`

Returns

Line width of the frame

See also

setLineWidth(), frameShadow(), lineWidth()

Definition at line 215 of file qwt_dial.cpp.

14.32.4.14 maxScaleArc() `double QwtDial::maxScaleArc ( ) const`

Returns

Upper limit of the scale arc

See also

setScaleArc()

Definition at line 647 of file qwt_dial.cpp.

14.32.4.15 minimumSizeHint() `QSize QwtDial::minimumSizeHint ( ) const [override], [virtual]`

Returns

Minimum size hint

See also

sizeHint()

Definition at line 702 of file qwt_dial.cpp.

14.32.4.16 minScaleArc() `double QwtDial::minScaleArc ( ) const`

Returns

Lower limit of the scale arc

See also

setScaleArc()

Definition at line 627 of file qwt_dial.cpp.

14.32.4.17 mode() `QwtDial::Mode QwtDial::mode ( ) const`

Returns

Mode of the dial.

See also

setMode(), origin(), setScaleArc(), value()

Definition at line 293 of file qwt_dial.cpp.

14.32.4.18 needle() [1/2] `QwtDialNeedle * QwtDial::needle ( )`

Returns

needle

See also

setNeedle()

Definition at line 547 of file qwt_dial.cpp.

14.32.4.19 needle() [2/2] const QwtDialNeedle * QwtDial::needle ( ) const

Returns

needle

See also

setNeedle()

Definition at line 538 of file qwt_dial.cpp.

14.32.4.20 origin() double QwtDial::origin ( ) const

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

Returns

Origin of the dial

See also

setOrigin()

Definition at line 674 of file qwt_dial.cpp.

14.32.4.21 paintEvent() void QwtDial::paintEvent (QPaintEvent * *event*) [override], [protected], [virtual]

Paint the dial

Parameters

event	Paint event

Definition at line 310 of file qwt_dial.cpp.

14.32.4.22 scaleChange() void QwtDial::scaleChange ( ) [override], [protected], [virtual]

Invalidate the internal caches and call QwtAbstractSlider::scaleChange()

Reimplemented from QwtAbstractSlider.

Definition at line 855 of file qwt_dial.cpp.

14.32.4.23 scaleDraw() [1/2] QwtRoundScaleDraw * QwtDial::scaleDraw ( )

Returns

the scale draw

Definition at line 553 of file qwt_dial.cpp.

14.32.4.24 scaleDraw() [2/2] const QwtRoundScaleDraw * QwtDial::scaleDraw ( ) const

Returns

the scale draw

Definition at line 559 of file qwt_dial.cpp.

14.32.4.25 scaleInnerRect() QRect QwtDial::scaleInnerRect ( ) const [virtual]

Returns

rectangle inside the scale

See also

setLineWidth(), boundingRect(), innerRect()

Definition at line 250 of file qwt_dial.cpp.

14.32.4.26 scrolledTo() double QwtDial::scrolledTo (const QPoint & *pos*) const [override], [protected], [virtual]

Determine the value for a new position of the slider handle.

Parameters

pos	Mouse position

Returns

Value for the mouse position

See also

isScrollPosition()

Implements QwtAbstractSlider.

Definition at line 751 of file qwt_dial.cpp.

14.32.4.27 setFrameShadow() `void QwtDial::setFrameShadow ( Shadow shadow )`

Sets the frame shadow value from the frame style.

Parameters

shadow	Frame shadow

See also

setLineWidth(), QFrame::setFrameShadow()

Definition at line 170 of file qwt_dial.cpp.

14.32.4.28 setLineWidth() `void QwtDial::setLineWidth ( int lineWidth )`

Sets the line width of the frame

Parameters

lineWidth	Line width

See also

setFrameShadow()

Definition at line 197 of file qwt_dial.cpp.

14.32.4.29 setMaxScaleArc() `void QwtDial::setMaxScaleArc ( double max )`

Set the upper limit for the scale arc

Parameters

max	Upper limit of the scale arc

See also

setScaleArc(), setMinScaleArc()

Definition at line 638 of file qwt_dial.cpp.

14.32.4.30 setMinScaleArc() `void QwtDial::setMinScaleArc ( double min )`

Set the lower limit for the scale arc

Parameters

min	Lower limit of the scale arc

See also

setScaleArc(), setMaxScaleArc()

Definition at line 618 of file qwt_dial.cpp.

14.32.4.31 setMode() `void QwtDial::setMode ( Mode mode )`

Change the mode of the dial.

Parameters

mode	New mode

In case of QwtDial::RotateNeedle 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()

Definition at line 278 of file qwt_dial.cpp.

14.32.4.32 setNeedle() void QwtDial::setNeedle (QwtDialNeedle * needle)

Set a needle for the dial

Parameters

needle	Needle

Warning

The needle will be deleted, when a different needle is set or in ~QwtDial()

Definition at line 522 of file qwt_dial.cpp.

14.32.4.33 setOrigin() void QwtDial::setOrigin (double origin) [virtual]

Change the origin.

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

Parameters

origin	New origin

See also

origin()

Definition at line 660 of file qwt_dial.cpp.

14.32.4.34 setScaleArc() void QwtDial::setScaleArc (double minArc, double maxArc)

Change the arc of the scale

Parameters

minArc	Lower limit
maxArc	Upper limit

See also

minScaleArc(), maxScaleArc()

Definition at line 588 of file qwt_dial.cpp.

14.32.4.35 setScaleDraw() `void QwtDial::setScaleDraw ( QwtRoundScaleDraw * scaleDraw )`

Set an individual scale draw

The motivation for setting a scale draw is often to overload QwtRoundScaleDraw::label() to return individual tick labels.

Parameters

scaleDraw	Scale draw

Warning

The previous scale draw is deleted

Definition at line 574 of file qwt_dial.cpp.

14.32.4.36 sizeHint() `QSize QwtDial::sizeHint ( ) const [override], [virtual]`

Returns

Size hint

See also

minimumSizeHint()

Definition at line 683 of file qwt_dial.cpp.

14.32.4.37 wheelEvent() `void QwtDial::wheelEvent ( QWheelEvent * event ) [override], [protected], [virtual]`

Wheel Event handler

Parameters

event	Wheel event

Reimplemented from QwtAbstractSlider.

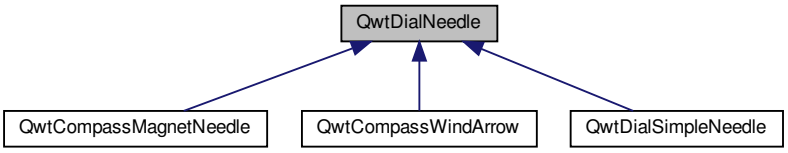
Definition at line 829 of file qwt_dial.cpp.

14.33 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

- QwtDialNeedle ()
  Constructor.
- virtual ~QwtDialNeedle ()
  Destructor.
- virtual void setPalette (const QPalette &)
- const QPalette & palette () const
- virtual void draw (QPainter *, const QPointF &center, double length, double direction, QPalette::ColorGroup=QPalette::Active) const

Protected Member Functions

- virtual void drawNeedle (QPainter *painter, double length, QPalette::ColorGroup colorGroup) const =0
  Draw the needle.
- virtual void drawKnob (QPainter *, double width, const QBrush &, bool sunken) const
  Draw the knob.

14.33.1 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.

See also

QwtDial, QwtCompass

Definition at line 27 of file qwt_dial_needle.h.

14.33.2 Member Function Documentation

14.33.2.1 draw()

```
void QwtDialNeedle::draw (
    QPainter * painter,
    const QPointF & center,
    double length,
    double direction,
    QPalette::ColorGroup colorGroup = QPalette::Active ) const  [virtual]
```

Draw the needle

Parameters

painter	Painter
center	Center of the dial, start position for the needle
length	Length of the needle
direction	Direction of the needle, in degrees counter clockwise
colorGroup	Color group, used for painting

Definition at line 219 of file qwt_dial_needle.cpp.

14.33.2.2 drawNeedle()

```
virtual void QwtDialNeedle::drawNeedle (
    QPainter * painter,
    double length,
    QPalette::ColorGroup colorGroup ) const  [protected], [pure virtual]
```

Draw the needle.

The origin of the needle is at position (0.0, 0.0) pointing in direction 0.0 (= east).

The painter is already initialized with translation and rotation.

Parameters

painter	Painter
length	Length of the needle
colorGroup	Color group, used for painting

See also

setPalette(), palette()

Implemented in QwtCompassWindArrow, QwtCompassMagnetNeedle, and QwtDialSimpleNeedle.

14.33.2.3 palette() `const QPalette & QwtDialNeedle::palette ( ) const`

Returns

the palette of the needle.

Definition at line 205 of file qwt_dial_needle.cpp.

14.33.2.4 setPalette() `void QwtDialNeedle::setPalette ( const QPalette & palette ) [virtual]`

Sets the palette for the needle.

Parameters

palette	New Palette

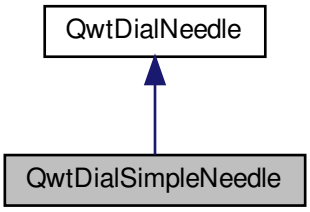
Definition at line 197 of file qwt_dial_needle.cpp.

14.34 QwtDialSimpleNeedle Class Reference

A needle for dial widgets.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtDialSimpleNeedle:



Public Types

- enum Style { Arrow , Ray }

  Style of the needle.

Public Member Functions

- QwtDialSimpleNeedle (Style, bool hasKnob=true, const QColor &mid=Qt::gray, const QColor &base=Qt::darkGray)
- void setWidth (double width)
- double width () const

Protected Member Functions

- virtual void drawNeedle (QPainter *, double length, QPalette::ColorGroup) const override

14.34.1 Detailed Description

A needle for dial widgets.

The following colors are used:

- QPalette::Mid
  Pointer
- QPalette::Base
  Knob

See also

QwtDial, QwtCompass

Definition at line 81 of file qwt_dial_needle.h.

14.34.2 Member Enumeration Documentation

14.34.2.1 Style `enum QwtDialSimpleNeedle::Style`

Style of the needle.

Enumerator

Arrow	Arrow.
Ray	A straight line from the center.

Definition at line 85 of file qwt_dial_needle.h.

14.34.3 Constructor & Destructor Documentation

14.34.3.1 QwtDialSimpleNeedle() `QwtDialSimpleNeedle::QwtDialSimpleNeedle (`

```
Style style,
bool hasKnob = true,
const QColor & mid = Qt::gray,
const QColor & base = Qt::darkGray )
```

Constructor

Parameters

style	Style
hasKnob	With/Without knob
mid	Middle color
base	Base color

Definition at line 273 of file qwt_dial_needle.cpp.

14.34.4 Member Function Documentation

14.34.4.1 drawNeedle() `void QwtDialSimpleNeedle::drawNeedle (`

```
QPainter * painter,
double length,
QPalette::ColorGroup colorGroup ) const  [override], [protected], [virtual]
```

Draw the needle

Parameters

painter	Painter
length	Length of the needle
colorGroup	Color group, used for painting

Implements QwtDialNeedle.

Definition at line 312 of file qwt_dial_needle.cpp.

14.34.4.2 setWidth() `void QwtDialSimpleNeedle::setWidth (`

```
double width )
```

Set the width of the needle

Parameters

width	Width

See also

width()

Definition at line 291 of file qwt_dial_needle.cpp.

14.34.4.3 width() `double QwtDialSimpleNeedle::width ( ) const`

Returns

the width of the needle

See also

setWidth()

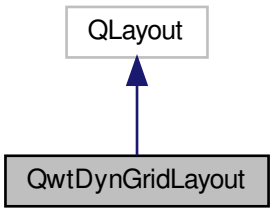
Definition at line 300 of file qwt_dial_needle.cpp.

14.35 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>`

Inheritance diagram for QwtDynGridLayout:



Public Member Functions

- QwtDynGridLayout (QWidget ∗, int margin=0, int spacing=-1)
- QwtDynGridLayout (int spacing=-1)
- virtual ∼QwtDynGridLayout ()

  Destructor.
- virtual void invalidate () override

  Invalidate all internal caches.
- void setMaxColumns (uint maxColumns)
- uint maxColumns () const

  Return the upper limit for the number of columns.
- uint numRows () const
- uint numColumns () const
- virtual void addItem (QLayoutItem ∗) override

  Add an item to the next free position.
- virtual QLayoutItem ∗ itemAt (int index) const override
- virtual QLayoutItem ∗ takeAt (int index) override
- virtual int count () const override
- void setExpandingDirections (Qt::Orientations)
- virtual Qt::Orientations expandingDirections () const override

  Returns whether this layout can make use of more space than sizeHint().
- QList< QRect > layoutItems (const QRect &, uint numColumns) const
- virtual int maxItemWidth () const
- virtual void setGeometry (const QRect &) override
- virtual bool hasHeightForWidth () const override
- virtual int heightForWidth (int) const override
- virtual QSize sizeHint () const override
- virtual bool isEmpty () const override
- uint itemCount () const
- virtual uint columnsForWidth (int width) const

  Calculate the number of columns for a given width.

Protected Member Functions

- void layoutGrid (uint numColumns, QVector< int > &rowHeight, QVector< int > &colWidth) const
- void stretchGrid (const QRect &rect, uint numColumns, QVector< int > &rowHeight, QVector< int > &col↩ Width) const

14.35.1 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 maxColumns()).

Definition at line 27 of file qwt_dyngrid_layout.h.

14.35.2 Constructor & Destructor Documentation

14.35.2.1 QwtDynGridLayout() [1/2]

```
QwtDynGridLayout::QwtDynGridLayout (
    QWidget * parent,
    int margin = 0,
    int spacing = -1 )  [explicit]
```

Parameters

parent	Parent widget
margin	Margin
spacing	Spacing

Definition at line 58 of file qwt_dyngrid_layout.cpp.

14.35.2.2 QwtDynGridLayout() [2/2] QwtDynGridLayout::QwtDynGridLayout (int *spacing* = *-1*) [explicit]

Parameters

spacing	Spacing

Definition at line 71 of file qwt_dyngrid_layout.cpp.

14.35.3 Member Function Documentation

14.35.3.1 addItem() void QwtDynGridLayout::addItem (QLayoutItem * *item*) [override], [virtual]

Add an item to the next free position.

Parameters

item	Layout item

Definition at line 128 of file qwt_dyngrid_layout.cpp.

14.35.3.2 columnsForWidth() uint QwtDynGridLayout::columnsForWidth (int *width*) const [virtual]

Calculate the number of columns for a given width.

The calculation tries to use as many columns as possible (limited by maxColumns())

Parameters

width	Available width for all columns

Returns

Number of columns for a given width

See also

maxColumns(), setMaxColumns()

Definition at line 256 of file qwt_dyngrid_layout.cpp.

14.35.3.3 count() `int QwtDynGridLayout::count ( ) const [override], [virtual]`

Returns

Number of items in the layout

Definition at line 182 of file qwt_dyngrid_layout.cpp.

14.35.3.4 expandingDirections() `Qt::Orientations QwtDynGridLayout::expandingDirections ( ) const [override], [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.

Returns

Orientations, where the layout expands

See also

setExpandingDirections()

Definition at line 211 of file qwt_dyngrid_layout.cpp.

14.35.3.5 hasHeightForWidth() `bool QwtDynGridLayout::hasHeightForWidth ( ) const [override], [virtual]`

Returns

true: QwtDynGridLayout implements heightForWidth().

See also

heightForWidth()

Definition at line 446 of file qwt_dyngrid_layout.cpp.

14.35.3.6 heightForWidth() `int QwtDynGridLayout::heightForWidth ( int width ) const [override], [virtual]`

Returns

The preferred height for this layout, given a width.

See also

hasHeightForWidth()

Definition at line 455 of file qwt_dyngrid_layout.cpp.

14.35.3.7 isEmpty() `bool QwtDynGridLayout::isEmpty ( ) const [override], [virtual]`

Returns

true if this layout is empty.

Definition at line 137 of file qwt_dyngrid_layout.cpp.

14.35.3.8 itemAt() `QLayoutItem * QwtDynGridLayout::itemAt ( int index ) const [override], [virtual]`

Find the item at a specific index

Parameters

index	Index

Returns

Item at a specific index

See also

takeAt()

Definition at line 157 of file qwt_dyngrid_layout.cpp.

14.35.3.9 itemCount() `uint QwtDynGridLayout::itemCount ( ) const`

Returns

number of layout items

Definition at line 145 of file qwt_dyngrid_layout.cpp.

14.35.3.10 layoutGrid()

```
void QwtDynGridLayout::layoutGrid (
    uint numColumns,
    QVector< int > & rowHeight,
    QVector< int > & colWidth ) const  [protected]
```

Calculate the dimensions for the columns and rows for a grid of numColumns columns.

Parameters

numColumns	Number of columns.
rowHeight	Array where to fill in the calculated row heights.
colWidth	Array where to fill in the calculated column widths.

Definition at line 419 of file qwt_dyngrid_layout.cpp.

14.35.3.11 layoutItems()

```
QList< QRect > QwtDynGridLayout::layoutItems (
    const QRect & rect,
    uint numColumns ) const
```

Calculate the geometries of the layout items for a layout with numColumns columns and a given rectangle.

Parameters

rect	Rect where to place the items
numColumns	Number of columns

Returns

item geometries

Definition at line 344 of file qwt_dyngrid_layout.cpp.

14.35.3.12 maxColumns()

```
uint QwtDynGridLayout::maxColumns ( ) const
```

Return the upper limit for the number of columns.

0 means unlimited, what is the default.

Returns

Upper limit for the number of columns

See also

setMaxColumns()

Definition at line 119 of file qwt_dyngrid_layout.cpp.

14.35.3.13 maxItemWidth() `int QwtDynGridLayout::maxItemWidth ( ) const [virtual]`

Returns

the maximum width of all layout items

Definition at line 316 of file qwt_dyngrid_layout.cpp.

14.35.3.14 numColumns() `uint QwtDynGridLayout::numColumns ( ) const`

Returns

Number of columns of the current layout.

See also

numRows()

Warning

The number of columns might change whenever the geometry changes

Definition at line 596 of file qwt_dyngrid_layout.cpp.

14.35.3.15 numRows() `uint QwtDynGridLayout::numRows ( ) const`

Returns

Number of rows of the current layout.

See also

numColumns()

Warning

The number of rows might change whenever the geometry changes

Definition at line 586 of file qwt_dyngrid_layout.cpp.

14.35.3.16 setExpandingDirections() `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

expanding	Or'd orientations

See also

expandingDirections()

Definition at line 196 of file qwt_dyngrid_layout.cpp.

14.35.3.17 setGeometry() `void QwtDynGridLayout::setGeometry ( const QRect & rect ) [override], [virtual]`

Reorganizes columns and rows and resizes managed items within a rectangle.

Parameters

rect	Layout geometry

Definition at line 222 of file qwt_dyngrid_layout.cpp.

14.35.3.18 setMaxColumns() `void QwtDynGridLayout::setMaxColumns ( uint maxColumns )`

Limit the number of columns.

Parameters

maxColumns	upper limit, 0 means unlimited

See also

maxColumns()

Definition at line 106 of file qwt_dyngrid_layout.cpp.

14.35.3.19 sizeHint() `QSize QwtDynGridLayout::sizeHint ( ) const [override], [virtual]`

Return the size hint. If maxColumns() $>$ 0 it is the size for a grid with maxColumns() columns, otherwise it is the size for a grid with only one row.

Returns

Size hint

See also

maxColumns(), setMaxColumns()

Definition at line 550 of file qwt_dyngrid_layout.cpp.

14.35.3.20 stretchGrid()

```
void QwtDynGridLayout::stretchGrid (
    const QRect & rect,
    uint numColumns,
    QVector< int > & rowHeight,
    QVector< 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.

Parameters

rect	Bounding rectangle
numColumns	Number of columns
rowHeight	Array to be filled with the calculated row heights
colWidth	Array to be filled with the calculated column widths

See also

setExpanding(), expanding()

Definition at line 491 of file qwt_dyngrid_layout.cpp.

14.35.3.21 takeAt()

```
QLayoutItem * QwtDynGridLayout::takeAt (
    int index )  [override], [virtual]
```

Find the item at a specific index and remove it from the layout

Parameters

index	Index

Returns

Layout item, removed from the layout

See also

itemAt()

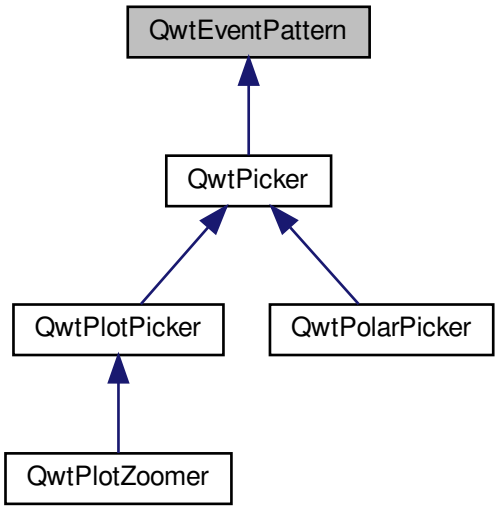
Definition at line 172 of file qwt_dyngrid_layout.cpp.

14.36 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 MousePatternCode {
  MouseSelect1 , MouseSelect2 , MouseSelect3 , MouseSelect4 ,
  MouseSelect5 , MouseSelect6 , MousePatternCount }

  Symbolic mouse input codes.
- enum KeyPatternCode {
  KeySelect1 , KeySelect2 , KeyAbort , KeyLeft ,
  KeyRight , KeyUp , KeyDown , KeyRedo ,
  KeyUndo , KeyHome , KeyPatternCount }

  Symbolic keyboard input codes.

Public Member Functions

- QwtEventPattern ()
- virtual ~QwtEventPattern ()

  Destructor.
- void initMousePattern (int numButtons)
- void initKeyPattern ()
- void setMousePattern (MousePatternCode, Qt::MouseButton button, Qt::KeyboardModifiers=Qt::NoModifier)
- void setKeyPattern (KeyPatternCode, int key, Qt::KeyboardModifiers modifiers=Qt::NoModifier)
- void setMousePattern (const QVector< MousePattern > &)

  Change the mouse event patterns.
- void setKeyPattern (const QVector< KeyPattern > &)

  Change the key event patterns.
- const QVector< MousePattern > & mousePattern () const
- const QVector< KeyPattern > & keyPattern () const
- QVector< MousePattern > & mousePattern ()
- QVector< KeyPattern > & keyPattern ()
- bool mouseMatch (MousePatternCode, const QMouseEvent *) const

  Compare a mouse event with an event pattern.
- bool keyMatch (KeyPatternCode, const QKeyEvent *) const

  Compare a key event with an event pattern.

Protected Member Functions

- virtual bool mouseMatch (const MousePattern &, const QMouseEvent *) const

  Compare a mouse event with an event pattern.
- virtual bool keyMatch (const KeyPattern &, const QKeyEvent *) const

  Compare a key event with an event pattern.

14.36.1 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

Definition at line 30 of file qwt_event_pattern.h.

14.36.2 Member Enumeration Documentation

14.36.2.1 KeyPatternCode `enum QwtEventPattern::KeyPatternCode`

Symbolic keyboard input codes.

Individual settings can be configured using setKeyPattern()

See also

setKeyPattern(), setMousePattern()

Enumerator

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.
KeyRedo	Qt::Key_Plus.
KeyUndo	Qt::Key_Minus.
KeyHome	Qt::Key_Escape.
KeyPatternCount	Number of key patterns.

Definition at line 112 of file qwt_event_pattern.h.

14.36.2.2 MousePatternCode `enum QwtEventPattern::MousePatternCode`

Symbolic mouse input codes.

QwtEventPattern implements 3 different settings for mice with 1, 2, or 3 buttons that can be activated using initMousePattern(). The default setting is for 3 button mice.

Individual settings can be configured using setMousePattern().

See also

initMousePattern(), setMousePattern(), setKeyPattern()

Enumerator

MouseSelect1	The default setting for 1, 2 and 3 button mice is: • Qt::LeftButton • Qt::LeftButton • Qt::LeftButton
MouseSelect2	The default setting for 1, 2 and 3 button mice is: • Qt::LeftButton + Qt::ControlModifier • Qt::RightButton • Qt::RightButton
MouseSelect3	The default setting for 1, 2 and 3 button mice is: • Qt::LeftButton + Qt::AltModifier • Qt::LeftButton + Qt::AltModifier • Qt::MidButton

Enumerator

MouseSelect4	The default setting for 1, 2 and 3 button mice is: • Qt::LeftButton + Qt::ShiftModifier • Qt::LeftButton + Qt::ShiftModifier • Qt::LeftButton + Qt::ShiftModifier
MouseSelect5	The default setting for 1, 2 and 3 button mice is: • Qt::LeftButton + Qt::ControlButton | Qt::ShiftModifier • Qt::RightButton + Qt::ShiftModifier • Qt::RightButton + Qt::ShiftModifier
MouseSelect6	The default setting for 1, 2 and 3 button mice is: • Qt::LeftButton + Qt::AltModifier + Qt::ShiftModifier • Qt::LeftButton + Qt::AltModifier | Qt::ShiftModifier • Qt::MidButton + Qt::ShiftModifier
MousePatternCount	Number of mouse patterns.

Definition at line 45 of file qwt_event_pattern.h.

14.36.3 Constructor & Destructor Documentation

14.36.3.1 QwtEventPattern() `QwtEventPattern::QwtEventPattern ( )`

Constructor

See also

MousePatternCode, KeyPatternCode

Definition at line 19 of file qwt_event_pattern.cpp.

14.36.4 Member Function Documentation

14.36.4.1 initKeyPattern() `void QwtEventPattern::initKeyPattern ( )`

Set default mouse patterns.

See also

KeyPatternCode

Definition at line 81 of file qwt_event_pattern.cpp.

14.36.4.2 initMousePattern() `void QwtEventPattern::initMousePattern ( int numButtons )`

Set default mouse patterns, depending on the number of mouse buttons

Parameters

numButtons	Number of mouse buttons (<= 3)

See also

MousePatternCode

Definition at line 38 of file qwt_event_pattern.cpp.

14.36.4.3 keyMatch() [1/2] `bool QwtEventPattern::keyMatch ( const KeyPattern & pattern, const QKeyEvent * event ) 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

pattern	Key event pattern
event	Key event

Returns

true if matches

See also

mouseMatch()

Definition at line 257 of file qwt_event_pattern.cpp.

14.36.4.4 keyMatch() [2/2] `bool QwtEventPattern::keyMatch ( KeyPatternCode code, const QKeyEvent * event ) 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

code	Index of the event pattern
event	Key event

Returns

true if matches

See also

mouseMatch()

Definition at line 234 of file qwt_event_pattern.cpp.

14.36.4.5 keyPattern() [1/2] QVector< QwtEventPattern::KeyPattern > & QwtEventPattern::keyPattern ( )

Returns

Key pattern

Definition at line 170 of file qwt_event_pattern.cpp.

14.36.4.6 keyPattern() [2/2] const QVector< QwtEventPattern::KeyPattern > & QwtEventPattern::keyPattern ( ) const

Returns

Key pattern

Definition at line 158 of file qwt_event_pattern.cpp.

14.36.4.7 mouseMatch() [1/2] bool QwtEventPattern::mouseMatch (const MousePattern & *pattern*, const QMouseEvent * *event*) 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

pattern	Mouse event pattern
event	Mouse event

Returns

true if matches

See also

keyMatch()

Definition at line 211 of file qwt_event_pattern.cpp.

14.36.4.8 mouseMatch() [2/2]

```
bool QwtEventPattern::mouseMatch (
        MousePatternCode code,
        const QMouseEvent * event ) 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

code	Index of the event pattern
event	Mouse event

Returns

true if matches

See also

keyMatch()

Definition at line 188 of file qwt_event_pattern.cpp.

14.36.4.9 mousePattern() [1/2]

```
QVector< QwtEventPattern::MousePattern > & QwtEventPattern↩
::mousePattern ( )
```

Returns

Mouse pattern

Definition at line 164 of file qwt_event_pattern.cpp.

14.36.4.10 mousePattern() [2/2] const QVector< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern ( ) const

Returns

Mouse pattern

Definition at line 151 of file qwt_event_pattern.cpp.

14.36.4.11 setKeyPattern() void QwtEventPattern::setKeyPattern (KeyPatternCode *pattern*, int *key*, Qt::KeyboardModifiers *modifiers* = *Qt::NoModifier*)

Change one key pattern

Parameters

pattern	Index of the pattern
key	Key
modifiers	Keyboard modifiers

See also

QKeyEvent

Definition at line 127 of file qwt_event_pattern.cpp.

14.36.4.12 setMousePattern() void QwtEventPattern::setMousePattern (MousePatternCode *pattern*, Qt::MouseButton *button*, Qt::KeyboardModifiers *modifiers* = *Qt::NoModifier*)

Change one mouse pattern

Parameters

pattern	Index of the pattern
button	Button
modifiers	Keyboard modifiers

See also

QMouseEvent

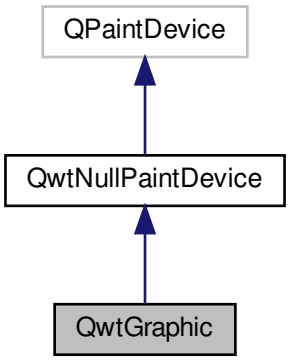
Definition at line 108 of file qwt_event_pattern.cpp.

14.37 QwtGraphic Class Reference

A paint device for scalable graphics.

`#include <qwt_graphic.h>`

Inheritance diagram for QwtGraphic:



Public Types

- enum RenderHint { RenderPensUnscaled = 0x1 }
- enum CommandType { VectorData = 1 << 0 , RasterData = 1 << 1 , Transformation = 1 << 2 }
- typedef QFlags< RenderHint > RenderHints
- typedef QFlags< CommandType > CommandTypes

Public Member Functions

- QwtGraphic ()

  Constructor.
- QwtGraphic (const QwtGraphic &)

  Copy constructor.
- virtual ~QwtGraphic ()

  Destructor.
- QwtGraphic & operator= (const QwtGraphic &)

  Assignment operator.
- void reset ()

  Clear all stored commands.
- bool isNull () const
- bool isEmpty () const
- CommandTypes commandTypes () const
- void render (QPainter *) const

  Replay all recorded painter commands.
- void render (QPainter *, const QSizeF &, Qt::AspectRatioMode=Qt::IgnoreAspectRatio) const

Replay all recorded painter commands.

- void render (QPainter ∗, const QPointF &, Qt::Alignment=Qt::AlignTop|Qt::AlignLeft) const

  Replay all recorded painter commands.
- void render (QPainter ∗, const QRectF &, Qt::AspectRatioMode=Qt::IgnoreAspectRatio) const

  Replay all recorded painter commands.
- QPixmap toPixmap (qreal devicePixelRatio=0.0) const

  Convert the graphic to a QPixmap.
- QPixmap toPixmap (const QSize &, Qt::AspectRatioMode=Qt::IgnoreAspectRatio, qreal devicePixel↩ Ratio=0.0) const

  Convert the graphic to a QPixmap.
- QImage toImage (qreal devicePixelRatio=0.0) const

  Convert the graphic to a QImage.
- QImage toImage (const QSize &, Qt::AspectRatioMode=Qt::IgnoreAspectRatio, qreal devicePixelRatio=0.0) const

  Convert the graphic to a QImage.
- QRectF scaledBoundingRect (qreal sx, qreal sy) const

  Calculate the target rectangle for scaling the graphic.
- QRectF boundingRect () const
- QRectF controlPointRect () const
- const QVector< QwtPainterCommand > & commands () const
- void setCommands (const QVector< QwtPainterCommand > &)

  Append paint commands.
- void setDefaultSize (const QSizeF &)

  Set a default size.
- QSizeF defaultSize () const

  Default size.
- qreal heightForWidth (qreal width) const
- qreal widthForHeight (qreal height) const
- void setRenderHint (RenderHint, bool on=true)
- bool testRenderHint (RenderHint) const
- RenderHints renderHints () const

Protected Member Functions

- virtual QSize sizeMetrics () const override
- virtual void drawPath (const QPainterPath &) override
- virtual void drawPixmap (const QRectF &, const QPixmap &, const QRectF &) override

  Store a pixmap command in the command list.
- virtual void drawImage (const QRectF &, const QImage &, const QRectF &, Qt::ImageConversionFlags) override

  Store a image command in the command list.
- virtual void updateState (const QPaintEngineState &) override

  Store a state command in the command list.

14.37.1 Detailed Description

A paint device for scalable graphics.

QwtGraphic is the representation of a graphic that is tailored for scalability. Like QPicture it will be initialized by QPainter operations and can be replayed later to any target paint device.

While the usual image representations QImage and QPixmap are not scalable Qt offers two paint devices, that might be candidates for representing a vector graphic:

- QPicture
  Unfortunately QPicture had been forgotten, when Qt4 introduced floating point based render engines. Its API is still on integers, what make it unusable for proper scaling.
- QSvgRenderer/QSvgGenerator
  Unfortunately QSvgRenderer hides to much information about its nodes in internal APIs, that are necessary for proper layout calculations. Also it is derived from QObject and can't be copied like QImage/QPixmap.

QwtGraphic maps all scalable drawing primitives to a QPainterPath and stores them together with the painter state changes (pen, brush, transformation ...) in a list of QwtPaintCommands. For being a complete QPaintDevice it also stores pixmaps or images, what is somehow against the idea of the class, because these objects can't be scaled without a loss in quality.

The main issue about scaling a QwtGraphic object are the pens used for drawing the outlines of the painter paths. While non cosmetic pens (QPen::isCosmetic()) are scaled with the same ratio as the path, cosmetic pens have a fixed width. A graphic might have paths with different pens - cosmetic and non-cosmetic.

QwtGraphic caches 2 different rectangles:

- control point rectangle
  The control point rectangle is the bounding rectangle of all control point rectangles of the painter paths, or the target rectangle of the pixmaps/images.
- bounding rectangle
  The bounding rectangle extends the control point rectangle by what is needed for rendering the outline with an unscaled pen.

Because the offset for drawing the outline depends on the shape of the painter path (the peak of a triangle is different than the flat side) scaling with a fixed aspect ratio always needs to be calculated from the control point rectangle.

See also

QwtPainterCommand

Definition at line 75 of file qwt_graphic.h.

14.37.2 Member Typedef Documentation

14.37.2.1 CommandTypes typedef QFlags<CommandType > QwtGraphic::CommandTypes

An ORed combination of CommandType values.

Definition at line 117 of file qwt_graphic.h.

14.37.2.2 RenderHints typedef QFlags<RenderHint > QwtGraphic::RenderHints

An ORed combination of RenderHint values.

Definition at line 99 of file qwt_graphic.h.

14.37.3 Member Enumeration Documentation

14.37.3.1 CommandType enum QwtGraphic::CommandType

Indicator if the graphic contains a specific type of painter command

See also

CommandTypes, commandTypes();

Enumerator

VectorData	The graphic contains scalable vector data.
RasterData	The graphic contains raster data (QPixmap or QImage)
Transformation	The graphic contains transformations beyond simple translations.

Definition at line 105 of file qwt_graphic.h.

14.37.3.2 RenderHint enum QwtGraphic::RenderHint

Hint how to render a graphic

See also

setRenderHint(), testRenderHint()

Enumerator

RenderPensUnscaled	When rendering a QwtGraphic a specific scaling between the controlPointRect() and the coordinates of the target rectangle is set up internally in render(). When RenderPensUnscaled is set this specific scaling is applied for the control points only, but not for the pens. All other painter transformations (set up by application code) are supposed to work like usual. **See also** render();

Definition at line 82 of file qwt_graphic.h.

14.37.4 Constructor & Destructor Documentation

14.37.4.1 QwtGraphic() [1/2] QwtGraphic::QwtGraphic ( )

Constructor.

Initializes a null graphic

See also

isNull()

Definition at line 355 of file qwt_graphic.cpp.

14.37.4.2 QwtGraphic() [2/2] QwtGraphic::QwtGraphic (const QwtGraphic & *other*)

Copy constructor.

Parameters

other	Source

See also

operator=()

Definition at line 367 of file qwt_graphic.cpp.

14.37.5 Member Function Documentation

14.37.5.1 boundingRect() `QRectF QwtGraphic::boundingRect ( ) const`

The bounding rectangle is the controlPointRect() extended by the areas needed for rendering the outlines with unscaled pens.

Returns

Bounding rectangle of the graphic

See also

controlPointRect(), scaledBoundingRect()

Definition at line 477 of file qwt_graphic.cpp.

14.37.5.2 commands() `const QVector< QwtPainterCommand > & QwtGraphic::commands ( ) const`

Returns

List of recorded paint commands

See also

setCommands()

Definition at line 1120 of file qwt_graphic.cpp.

14.37.5.3 commandTypes() `QwtGraphic::CommandTypes QwtGraphic::commandTypes ( ) const`

Returns

Types of painter commands being used

Definition at line 430 of file qwt_graphic.cpp.

14.37.5.4 controlPointRect() `QRectF QwtGraphic::controlPointRect ( ) const`

The control point rectangle is the bounding rectangle of all control points of the paths and the target rectangles of the images/pixmaps.

Returns

Control point rectangle

See also

boundingRect(), scaledBoundingRect()

Definition at line 493 of file qwt_graphic.cpp.

14.37.5.5 defaultSize() `QSizeF QwtGraphic::defaultSize ( ) const`

Default size.

When a non empty size has been assigned by setDefaultSize() this size will be returned. Otherwise the default size is the size of the bounding rectangle.

The default size is used in all methods rendering the graphic, where no size is explicitly specified.

Returns

Default size

See also

setDefaultSize(), boundingRect()

Definition at line 574 of file qwt_graphic.cpp.

14.37.5.6 drawImage()

```
void QwtGraphic::drawImage (
    const QRectF & rect,
    const QImage & image,
    const QRectF & subRect,
    Qt::ImageConversionFlags flags )  [override], [protected], [virtual]
```

Store a image command in the command list.

Parameters

rect	target rectangle
image	Image to be painted
subRect	Reactangle of the pixmap to be painted
flags	Image conversion flags

See also

QPaintEngine::drawImage()

Reimplemented from QwtNullPaintDevice.

Definition at line 1048 of file qwt_graphic.cpp.

14.37.5.7 drawPath()

```
void QwtGraphic::drawPath (
    const QPainterPath & path )  [override], [protected], [virtual]
```

Store a path command in the command list

Parameters

path	Painter path

See also

QPaintEngine::drawPath()

Reimplemented from QwtNullPaintDevice.

Definition at line 984 of file qwt_graphic.cpp.

14.37.5.8 drawPixmap()

```
void QwtGraphic::drawPixmap (
        const QRectF & rect,
        const QPixmap & pixmap,
        const QRectF & subRect ) [override], [protected], [virtual]
```

Store a pixmap command in the command list.

Parameters

rect	target rectangle
pixmap	Pixmap to be painted
subRect	Reactangle of the pixmap to be painted

See also

QPaintEngine::drawPixmap()

Reimplemented from QwtNullPaintDevice.

Definition at line 1023 of file qwt_graphic.cpp.

14.37.5.9 heightForWidth()

```
qreal QwtGraphic::heightForWidth (
        qreal width ) const
```

Find the height for a given width

The height is calculated using the aspect ratio of defaultSize().

Parameters

width	Width

Returns

Calculated height

See also

defaultSize()

Definition at line 592 of file qwt_graphic.cpp.

14.37.5.10 isEmpty() `bool QwtGraphic::isEmpty ( ) const`

Returns

True, when the bounding rectangle is empty

See also

boundingRect(), isNull()

Definition at line 422 of file qwt_graphic.cpp.

14.37.5.11 isNull() `bool QwtGraphic::isNull ( ) const`

Returns

True, when no painter commands have been stored

See also

isEmpty(), commands()

Definition at line 413 of file qwt_graphic.cpp.

14.37.5.12 operator=() `QwtGraphic & QwtGraphic::operator= ( const QwtGraphic & other )`

Assignment operator.

Parameters

other	Source

Returns

A reference of this object

Definition at line 385 of file qwt_graphic.cpp.

14.37.5.13 render() [1/4] void QwtGraphic::render (QPainter * *painter*) const

Replay all recorded painter commands.

Parameters

painter	Qt painter

Definition at line 624 of file qwt_graphic.cpp.

14.37.5.14 render() [2/4] void QwtGraphic::render (QPainter * *painter*, const QPointF & *pos*, Qt::Alignment *alignment* = *Qt::AlignTop* | *Qt::AlignLeft*) const

Replay all recorded painter commands.

The graphic is scaled to the defaultSize() and aligned to a position.

Parameters

painter	Qt painter
pos	Reference point, where to render
alignment	Flags how to align the target rectangle to pos.

Definition at line 762 of file qwt_graphic.cpp.

14.37.5.15 render() [3/4] void QwtGraphic::render (QPainter * *painter*, const QRectF & *rect*, Qt::AspectRatioMode *aspectRatioMode* = *Qt::IgnoreAspectRatio*) const

Replay all recorded painter commands.

The graphic is scaled to fit into the given rectangle

Parameters

painter	Qt painter
rect	Rectangle for the scaled graphic
aspectRatioMode	Mode how to scale - See Qt::AspectRatioMode

Definition at line 676 of file qwt_graphic.cpp.

14.37.5.16 render() [4/4]

```
void QwtGraphic::render (
    QPainter * painter,
    const QSizeF & size,
    Qt::AspectRatioMode aspectRatioMode = Qt::IgnoreAspectRatio ) const
```

Replay all recorded painter commands.

The graphic is scaled to fit into the rectangle of the given size starting at (0, 0).

Parameters

painter	Qt painter
size	Size for the scaled graphic
aspectRatioMode	Mode how to scale - See Qt::AspectRatioMode

Definition at line 660 of file qwt_graphic.cpp.

14.37.5.17 renderHints()

```
QwtGraphic::RenderHints QwtGraphic::renderHints ( ) const
```

Returns

Render hints

Definition at line 464 of file qwt_graphic.cpp.

14.37.5.18 reset()

```
void QwtGraphic::reset ( )
```

Clear all stored commands.

See also

isNull()

Definition at line 397 of file qwt_graphic.cpp.

14.37.5.19 scaledBoundingRect()

```
QRectF QwtGraphic::scaledBoundingRect (
    qreal sx,
    qreal sy ) const
```

Calculate the target rectangle for scaling the graphic.

Parameters

sx	Horizontal scaling factor
sy	Vertical scaling factor

Note

In case of paths that are painted with a cosmetic pen (see QPen::isCosmetic()) the target rectangle is different to multiplying the bounding rectangle.

Returns

Scaled bounding rectangle

See also

boundingRect(), controlPointRect()

Definition at line 514 of file qwt_graphic.cpp.

14.37.5.20 setCommands() `void QwtGraphic::setCommands ( const QVector< QwtPainterCommand > & commands )`

Append paint commands.

Parameters

commands	Paint commands

See also

commands()

Definition at line 1131 of file qwt_graphic.cpp.

14.37.5.21 setDefaultSize() `void QwtGraphic::setDefaultSize ( const QSizeF & size )`

Set a default size.

The default size is used in all methods rendering the graphic, where no size is explicitly specified. Assigning an empty size means, that the default size will be calculated from the bounding rectangle.

The default setting is an empty size.

Parameters

size	Default size

See also

defaultSize(), boundingRect()

Definition at line 553 of file qwt_graphic.cpp.

14.37.5.22 setRenderHint()

```
void QwtGraphic::setRenderHint (
    RenderHint hint,
    bool on = true )
```

Toggle an render hint

Parameters

hint	Render hint
on	true/false

See also

testRenderHint(), RenderHint

Definition at line 443 of file qwt_graphic.cpp.

14.37.5.23 sizeMetrics()

```
QSize QwtGraphic::sizeMetrics ( ) const  [override], [protected], [virtual]
```

Returns

Ceiled defaultSize()

Implements QwtNullPaintDevice.

Definition at line 533 of file qwt_graphic.cpp.

14.37.5.24 testRenderHint()

```
bool QwtGraphic::testRenderHint (
    RenderHint hint ) const
```

Test a render hint

Parameters

hint	Render hint

Returns

true/false

See also

setRenderHint(), RenderHint

Definition at line 458 of file qwt_graphic.cpp.

14.37.5.25 toImage() [1/2]

```
QImage QwtGraphic::toImage (
    const QSize & size,
    Qt::AspectRatioMode aspectRatioMode = Qt::IgnoreAspectRatio,
    qreal devicePixelRatio = 0.0 ) const
```

Convert the graphic to a QImage.

All pixels of the image get initialized by 0 (transparent) before the graphic is scaled and rendered on it.

The format of the image is QImage::Format_ARGB32_Premultiplied.

Parameters

size	Size of the image (will be multiplied by the devicePixelRatio)
aspectRatioMode	Aspect ratio how to scale the graphic
devicePixelRatio	Device pixel ratio for the image. If devicePixelRatio <= 0.0 the pixmap is initialized with the system default.

Returns

The graphic as image

See also

toPixmap(), render()

Definition at line 900 of file qwt_graphic.cpp.

14.37.5.26 toImage() [2/2]

```
QImage QwtGraphic::toImage (
    qreal devicePixelRatio = 0.0 ) const
```

Convert the graphic to a QImage.

All pixels of the image get initialized by 0 (transparent) before the graphic is scaled and rendered on it.

The format of the image is QImage::Format_ARGB32_Premultiplied.

The size of the image is the default size (ceiled to integers) of the graphic multiplied by the devicePixelRatio.

Parameters

devicePixelRatio	Device pixel ratio for the image. If devicePixelRatio <= 0.0 the pixmap is initialized with the system default.

Returns

The graphic as image in default size

See also

defaultSize(), toPixmap(), render()

Definition at line 943 of file qwt_graphic.cpp.

14.37.5.27 toPixmap() [1/2]

```
QPixmap QwtGraphic::toPixmap (
    const QSize & size,
    Qt::AspectRatioMode aspectRatioMode = Qt::IgnoreAspectRatio,
    qreal devicePixelRatio = 0.0 ) const
```

Convert the graphic to a QPixmap.

All pixels of the pixmap get initialized by Qt::transparent before the graphic is scaled and rendered on it.

Parameters

size	Size of the image
aspectRatioMode	Aspect ratio how to scale the graphic
devicePixelRatio	Device pixel ratio for the pixmap. If devicePixelRatio <= 0.0 the pixmap is initialized with the system default.

Returns

The graphic as pixmap

See also

toImage(), render()

Definition at line 859 of file qwt_graphic.cpp.

14.37.5.28 toPixmap() [2/2]

```
QPixmap QwtGraphic::toPixmap (
    qreal devicePixelRatio = 0.0 ) const
```

Convert the graphic to a QPixmap.

All pixels of the pixmap get initialized by Qt::transparent before the graphic is scaled and rendered on it.

The size of the pixmap is the default size (ceiled to integers) of the graphic.

Parameters

devicePixelRatio	Device pixel ratio for the pixmap. If devicePixelRatio <= 0.0 the pixmap is initialized with the system default.

Returns

The graphic as pixmap in default size

See also

defaultSize(), toImage(), render()

Definition at line 812 of file qwt_graphic.cpp.

14.37.5.29 updateState() `void QwtGraphic::updateState ( const QPaintEngineState & state ) [override], [protected], [virtual]`

Store a state command in the command list.

Parameters

state	State to be stored

See also

QPaintEngine::updateState()

Reimplemented from QwtNullPaintDevice.

Definition at line 1070 of file qwt_graphic.cpp.

14.37.5.30 widthForHeight() `qreal QwtGraphic::widthForHeight ( qreal height ) const`

Find the width for a given height

The width is calculated using the aspect ratio of defaultSize().

Parameters

height	Height

Returns

Calculated width

See also

defaultSize()

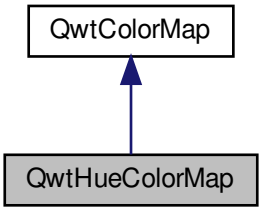
Definition at line 611 of file qwt_graphic.cpp.

14.38 QwtHueColorMap Class Reference

QwtHueColorMap varies the hue value of the HSV color model.

`#include <qwt_color_map.h>`

Inheritance diagram for QwtHueColorMap:



Public Member Functions

- QwtHueColorMap (QwtColorMap::Format=QwtColorMap::RGB)
  Constructor.
- virtual ~QwtHueColorMap ()
  Destructor.
- void setHueInterval (int hue1, int hue2)
- void setSaturation (int saturation)
  Set the the saturation coordinate.
- void setValue (int value)
  Set the the value coordinate.
- void setAlpha (int alpha)
  Set the the alpha coordinate.
- int hue1 () const
- int hue2 () const
- int saturation () const
- int value () const
- int alpha () const
- virtual QRgb rgb (const QwtInterval &, double value) const override

Additional Inherited Members

14.38.1 Detailed Description

QwtHueColorMap varies the hue value of the HSV color model.

QwtHueColorMap can be used to set up a color map easily, that runs cyclic over all colors. Each cycle has 360 different steps.

The values for value and saturation are in the range of 0 to 255 and doesn't depend on the data value to be mapped.

See also

QwtSaturationValueColorMap

Definition at line 180 of file qwt_color_map.h.

14.38.2 Constructor & Destructor Documentation

14.38.2.1 QwtHueColorMap() `QwtHueColorMap::QwtHueColorMap ( QwtColorMap::Format format = QwtColorMap::RGB ) [explicit]`

Constructor.

The hue interval is initialized by 0 to 359. All other coordinates are set to 255.

Parameters

format	Format of the color map

See also

setHueInterval(), setSaturation(), setValue(), setValue()

Definition at line 723 of file qwt_color_map.cpp.

14.38.3 Member Function Documentation

14.38.3.1 alpha() `int QwtHueColorMap::alpha ( ) const`

Returns

Alpha coordinate

See also

setAlpha()

Definition at line 856 of file qwt_color_map.cpp.

14.38.3.2 hue1() `int QwtHueColorMap::hue1 ( ) const`

Returns

First hue coordinate

See also

setHueInterval()

Definition at line 820 of file qwt_color_map.cpp.

14.38.3.3 hue2() `int QwtHueColorMap::hue2 ( ) const`

Returns

Second hue coordinate

See also

setHueInterval()

Definition at line 829 of file qwt_color_map.cpp.

14.38.3.4 rgb()

```
QRgb QwtHueColorMap::rgb (
    const QwtInterval & interval,
    double value ) const  [override], [virtual]
```

Map a value of a given interval into a RGB value

Parameters

interval	Range for all values
value	Value to map into a RGB value

Returns

RGB value for value

Implements QwtColorMap.

Definition at line 869 of file qwt_color_map.cpp.

14.38.3.5 saturation() `int QwtHueColorMap::saturation ( ) const`

Returns

Saturation coordinate

See also

setSaturation()

Definition at line 838 of file qwt_color_map.cpp.

14.38.3.6 setAlpha() `void QwtHueColorMap::setAlpha ( int alpha )`

Set the the alpha coordinate.

alpha needs to be in the range 0 to 255, where 255 means opaque and 0 means transparent.

Parameters

alpha	Alpha coordinate

See also

alpha()

Definition at line 805 of file qwt_color_map.cpp.

14.38.3.7 setHueInterval() `void QwtHueColorMap::setHueInterval ( int hue1, int hue2 )`

Set the interval for the hue coordinate

hue1/hue2 need to be positive number and can be $>$ 360 to define cycles. F.e. 420 to 240 defines a map yellow/red/magenta/blue.

Parameters

hue1	First hue coordinate
hue2	Second hue coordinate

See also

hue1(), hue2()

Definition at line 746 of file qwt_color_map.cpp.

14.38.3.8 setSaturation() `void QwtHueColorMap::setSaturation ( int saturation )`

Set the the saturation coordinate.

saturation needs to be in the range 0 to 255,

Parameters

saturation	Saturation coordinate

See also

saturation()

Definition at line 764 of file qwt_color_map.cpp.

14.38.3.9 setValue() `void QwtHueColorMap::setValue ( int value )`

Set the the value coordinate.

value needs to be in the range 0 to 255,

Parameters

value	Value coordinate

See also

value()

Definition at line 784 of file qwt_color_map.cpp.

14.38.3.10 value() `int QwtHueColorMap::value ( ) const`

Returns

Value coordinate

See also

setValue()

Definition at line 847 of file qwt_color_map.cpp.

14.39 QwtInterval Class Reference

A class representing an interval.

`#include <qwt_interval.h>`

Public Types

- enum BorderFlag { IncludeBorders = 0x00 , ExcludeMinimum = 0x01 , ExcludeMaximum = 0x02 , ExcludeBorders = ExcludeMinimum | ExcludeMaximum }
- typedef QFlags< BorderFlag > BorderFlags

  Border flags.

Public Member Functions

- QwtInterval ()

  Default Constructor.
- QwtInterval (double minValue, double maxValue, BorderFlags=IncludeBorders)
- void setInterval (double minValue, double maxValue, BorderFlags=IncludeBorders)
- QwtInterval normalized () const

  Normalize the limits of the interval.
- QwtInterval inverted () const
- QwtInterval limited (double lowerBound, double upperBound) const
- bool operator== (const QwtInterval &) const

  Compare two intervals.
- bool operator!= (const QwtInterval &) const

  Compare two intervals.
- void setBorderFlags (BorderFlags)
- BorderFlags borderFlags () const
- double minValue () const
- double maxValue () const
- double width () const

  Return the width of an interval.
- long double widthL () const

  Return the width of an interval as long double.
- void setMinValue (double)
- void setMaxValue (double)
- bool contains (double value) const

- bool contains (const QwtInterval &) const
- bool intersects (const QwtInterval &) const

  Test if two intervals overlap.
- QwtInterval intersect (const QwtInterval &) const

  Intersect 2 intervals.
- QwtInterval unite (const QwtInterval &) const

  Unite 2 intervals.
- QwtInterval operator| (const QwtInterval &) const
- QwtInterval operator& (const QwtInterval &) const

  Intersection of two intervals.
- QwtInterval & operator|= (const QwtInterval &)

  Unite this interval with the given interval.
- QwtInterval & operator&= (const QwtInterval &)

  Intersect this interval with the given interval.
- QwtInterval extend (double value) const

  Extend the interval.
- QwtInterval operator| (double) const
- QwtInterval & operator|= (double)
- bool isValid () const
- bool isNull () const
- void invalidate ()
- QwtInterval symmetrize (double value) const

14.39.1 Detailed Description

A class representing an interval.

The interval is represented by 2 doubles, the lower and the upper limit.

Definition at line 22 of file qwt_interval.h.

14.39.2 Member Typedef Documentation

14.39.2.1 BorderFlags `typedef QFlags<BorderFlag > QwtInterval::BorderFlags`

Border flags.

An ORed combination of BorderFlag values.

Definition at line 45 of file qwt_interval.h.

14.39.3 Member Enumeration Documentation

14.39.3.1 BorderFlag `enum QwtInterval::BorderFlag`

Flag indicating if a border is included or excluded

See also

setBorderFlags(), borderFlags()

Enumerator

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.

Definition at line 29 of file qwt_interval.h.

14.39.4 Constructor & Destructor Documentation

14.39.4.1 QwtInterval() [1/2] QwtInterval::QwtInterval ( ) [inline]

Default Constructor.

Creates an invalid interval [0.0, -1.0]

See also

setInterval(), isValid()

Definition at line 112 of file qwt_interval.h.

14.39.4.2 QwtInterval() [2/2] QwtInterval::QwtInterval (

```
double minValue,
double maxValue,
BorderFlags borderFlags = IncludeBorders ) [inline]
```

Constructor

Build an interval with from min/max values

Parameters

minValue	Minimum value
maxValue	Maximum value
borderFlags	Include/Exclude borders

Definition at line 128 of file qwt_interval.h.

14.39.5 Member Function Documentation

14.39.5.1 borderFlags() `QwtInterval::BorderFlags QwtInterval::borderFlags ( ) const [inline]`

Returns

Border flags

See also

setBorderFlags()

Definition at line 166 of file qwt_interval.h.

14.39.5.2 contains() `[1/2]` `bool QwtInterval::contains ( const QwtInterval & interval ) const`

Test if an interval is inside an interval

Parameters

interval	Interval

Returns

true, if interval lies inside the boundaries

Definition at line 90 of file qwt_interval.cpp.

14.39.5.3 contains() `[2/2]` `bool QwtInterval::contains ( double value ) const`

Test if a value is inside an interval

Parameters

value	Value

Returns

true, if value lies inside the boundaries

Definition at line 67 of file qwt_interval.cpp.

14.39.5.4 extend() QwtInterval QwtInterval::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

value	Value

Returns

extended interval

See also

isValid()

Definition at line 363 of file qwt_interval.cpp.

14.39.5.5 intersect() QwtInterval QwtInterval::intersect (const QwtInterval & *other*) const

Intersect 2 intervals.

Parameters

other	Interval to be intersect with

Returns

Intersection

Definition at line 186 of file qwt_interval.cpp.

14.39.5.6 intersects() bool QwtInterval::intersects (const QwtInterval & *other*) const

Test if two intervals overlap.

Parameters

other	Interval

Returns

True, when the intervals are intersecting

Definition at line 277 of file qwt_interval.cpp.

14.39.5.7 invalidate() `void QwtInterval::invalidate ( ) [inline]`

Invalidate the interval

The limits are set to interval [0.0, -1.0]

See also

isValid()

Definition at line 325 of file qwt_interval.h.

14.39.5.8 inverted() `QwtInterval QwtInterval::inverted ( ) const`

Invert the limits of the interval

Returns

Inverted interval

See also

normalized()

Definition at line 48 of file qwt_interval.cpp.

14.39.5.9 isNull() `bool QwtInterval::isNull ( ) const [inline]`

Returns

true, if isValid() && (minValue() >= maxValue())

Definition at line 314 of file qwt_interval.h.

14.39.5.10 isValid() `bool QwtInterval::isValid ( ) const [inline]`

A interval is valid when minValue() <= maxValue(). In case of QwtInterval::ExcludeBorders it is true when minValue() < maxValue()

Returns

True, when the interval is valid

Definition at line 210 of file qwt_interval.h.

14.39.5.11 limited()

```
QwtInterval QwtInterval::limited (
        double lowerBound,
        double upperBound ) const
```

Limit the interval, keeping the border modes

Parameters

lowerBound	Lower limit
upperBound	Upper limit

Returns

Limited interval

Definition at line 336 of file qwt_interval.cpp.

14.39.5.12 maxValue() `double QwtInterval::maxValue ( ) const [inline]`

Returns

Upper limit of the interval

Definition at line 198 of file qwt_interval.h.

14.39.5.13 minValue() `double QwtInterval::minValue ( ) const [inline]`

Returns

Lower limit of the interval

Definition at line 192 of file qwt_interval.h.

14.39.5.14 normalized() `QwtInterval QwtInterval::normalized ( ) const`

Normalize the limits of the interval.

If maxValue() $<$ minValue() the limits will be inverted.

Returns

Normalized interval

See also

isValid(), inverted()

Definition at line 29 of file qwt_interval.cpp.

14.39.5.15 operator"!=() `bool QwtInterval::operator!= ( const QwtInterval & other ) const [inline]`

Compare two intervals.

Parameters

other	Interval to compare with

Returns

True, when this and other are not equal

Definition at line 296 of file qwt_interval.h.

14.39.5.16 operator&() `QwtInterval QwtInterval::operator& ( const QwtInterval & other ) const [inline]`

Intersection of two intervals.

Parameters

other	Interval to intersect with

Returns

Intersection of this and other

See also

intersect()

Definition at line 258 of file qwt_interval.h.

14.39.5.17 operator&=() `QwtInterval & QwtInterval::operator&= ( const QwtInterval & other )`

Intersect this interval with the given interval.

Parameters

other	Interval to be intersected with

Returns

This interval

Definition at line 265 of file qwt_interval.cpp.

14.39.5.18 operator==() bool QwtInterval::operator== (const QwtInterval & *other*) const [inline]

Compare two intervals.

Parameters

other	Interval to compare with

Returns

True, when this and other are equal

Definition at line 284 of file qwt_interval.h.

14.39.5.19 operator"|() [1/2] QwtInterval QwtInterval::operator| (const QwtInterval & *other*) const [inline]

Union of two intervals

Parameters

other	Interval to unite with

Returns

Union of this and other

See also

unite()

Definition at line 272 of file qwt_interval.h.

14.39.5.20 operator"|() [2/2] QwtInterval QwtInterval::operator| (double *value*) const [inline]

Extend an interval

Parameters

value	Value

Returns

Extended interval

See also

extend()

Definition at line 308 of file qwt_interval.h.

14.39.5.21 operator"|=() [1/2] QwtInterval & QwtInterval::operator|= (const QwtInterval & *other*)

Unite this interval with the given interval.

Parameters

other	Interval to be united with

Returns

This interval

Definition at line 253 of file qwt_interval.cpp.

14.39.5.22 operator"|=() [2/2] QwtInterval & QwtInterval::operator|= (double *value*)

Extend an interval

Parameters

value	Value

Returns

Reference of the extended interval

See also

extend()

Definition at line 380 of file qwt_interval.cpp.

14.39.5.23 setBorderFlags() void QwtInterval::setBorderFlags (BorderFlags *borderFlags*) [inline]

Change the border flags

Parameters

borderFlags	Or'd BorderMode flags

See also

borderFlags()

Definition at line 157 of file qwt_interval.h.

14.39.5.24 setInterval()

```
void QwtInterval::setInterval (
    double minValue,
    double maxValue,
    BorderFlags borderFlags = IncludeBorders )  [inline]
```

Assign the limits of the interval

Parameters

minValue	Minimum value
maxValue	Maximum value
borderFlags	Include/Exclude borders

Definition at line 143 of file qwt_interval.h.

14.39.5.25 setMaxValue()

```
void QwtInterval::setMaxValue (
    double maxValue )  [inline]
```

Assign the upper limit of the interval

Parameters

maxValue	Maximum value

Definition at line 186 of file qwt_interval.h.

14.39.5.26 setMinValue()

```
void QwtInterval::setMinValue (
    double minValue )  [inline]
```

Assign the lower limit of the interval

Parameters

minValue	Minimum value

Definition at line 176 of file qwt_interval.h.

14.39.5.27 symmetrize() `QwtInterval QwtInterval::symmetrize ( double value ) const`

Adjust the limit that is closer to value, so that value becomes the center of the interval.

Parameters

value	Center

Returns

Interval with value as center

Definition at line 317 of file qwt_interval.cpp.

14.39.5.28 width() `double QwtInterval::width ( ) const [inline]`

Return the width of an interval.

The width of invalid intervals is 0.0, otherwise the result is maxValue() - minValue().

Returns

Interval width

See also

isValid()

Definition at line 227 of file qwt_interval.h.

14.39.5.29 widthL() `long double QwtInterval::widthL ( ) const [inline]`

Return the width of an interval as long double.

The width of invalid intervals is 0.0, otherwise the result is maxValue() - minValue().

Returns

Interval width

See also

isValid()

Definition at line 241 of file qwt_interval.h.

14.40 QwtIntervalSample Class Reference

A sample of the types (x1-x2, y) or (x, y1-y2)

`#include <qwt_samples.h>`

Public Member Functions

- QwtIntervalSample ()
- QwtIntervalSample (double, const QwtInterval &)

  Constructor.
- QwtIntervalSample (double value, double min, double max)

  Constructor.
- bool operator== (const QwtIntervalSample &) const

  Compare operator.
- bool operator!= (const QwtIntervalSample &) const

  Compare operator.

Public Attributes

- double value

  Value.
- QwtInterval interval

  Interval.

14.40.1 Detailed Description

A sample of the types (x1-x2, y) or (x, y1-y2)

Definition at line 20 of file qwt_samples.h.

14.40.2 Constructor & Destructor Documentation

14.40.2.1 QwtIntervalSample() `QwtIntervalSample::QwtIntervalSample ( ) [inline]`

Constructor The value is set to 0.0, the interval is invalid

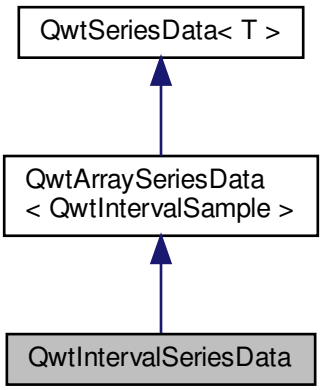
Definition at line 41 of file qwt_samples.h.

14.41 QwtIntervalSeriesData Class Reference

Interface for iterating over an array of intervals.

`#include <qwt_series_data.h>`

Inheritance diagram for QwtIntervalSeriesData:



Public Member Functions

- QwtIntervalSeriesData (const QVector< QwtIntervalSample > &=QVector< QwtIntervalSample >())
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.

Additional Inherited Members

14.41.1 Detailed Description

Interface for iterating over an array of intervals.

Definition at line 229 of file qwt_series_data.h.

14.41.2 Constructor & Destructor Documentation

14.41.2.1 QwtIntervalSeriesData() `QwtIntervalSeriesData::QwtIntervalSeriesData ( const QVector< QwtIntervalSample > & samples = QVector< QwtIntervalSample >( ) )`

Constructor

Parameters

samples	Samples

Definition at line 301 of file qwt_series_data.cpp.

14.41.3 Member Function Documentation

14.41.3.1 boundingRect() `QRectF QwtIntervalSeriesData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

Implements QwtSeriesData< T >.

Definition at line 315 of file qwt_series_data.cpp.

14.42 QwtIntervalSymbol Class Reference

A drawing primitive for displaying an interval like an error bar.

`#include <qwt_interval_symbol.h>`

Public Types

- enum Style { NoSymbol = -1 , Bar , Box , UserSymbol = 1000 }
  Symbol style.

Public Member Functions

- QwtIntervalSymbol (Style=NoSymbol)
- QwtIntervalSymbol (const QwtIntervalSymbol &)

  Copy constructor.
- virtual ~QwtIntervalSymbol ()

  Destructor.
- QwtIntervalSymbol & operator= (const QwtIntervalSymbol &)

  Assignment operator.
- bool operator== (const QwtIntervalSymbol &) const

  Compare two symbols.
- bool operator!= (const QwtIntervalSymbol &) const

  Compare two symbols.
- void setWidth (int)
- int width () const
- void setBrush (const QBrush &)

  Assign a brush.
- const QBrush & brush () const
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)
- const QPen & pen () const
- void setStyle (Style)
- Style style () const
- virtual void draw (QPainter *, Qt::Orientation, const QPointF &from, const QPointF &to) const

14.42.1 Detailed Description

A drawing primitive for displaying an interval like an error bar.

See also

QwtPlotIntervalCurve

Definition at line 27 of file qwt_interval_symbol.h.

14.42.2 Member Enumeration Documentation

14.42.2.1 Style `enum QwtIntervalSymbol::Style`

Symbol style.

Enumerator

NoSymbol	No Style. The symbol cannot be drawn.
Bar	The symbol displays a line with caps at the beginning/end. The size of the caps depends on the symbol width().
Box	The symbol displays a plain rectangle using pen() and brush(). The size of the rectangle depends on the translated interval and the width(),
UserSymbol	Styles >= UserSymbol are reserved for derived classes of QwtIntervalSymbol that overload draw() with additional application specific symbol types.

Definition at line 31 of file qwt_interval_symbol.h.

14.42.3 Constructor & Destructor Documentation

14.42.3.1 QwtIntervalSymbol() `QwtIntervalSymbol::QwtIntervalSymbol ( Style style = NoSymbol ) [explicit]`

Constructor

Parameters

style	Style of the symbol

See also

setStyle(), style(), Style

Definition at line 47 of file qwt_interval_symbol.cpp.

14.42.4 Member Function Documentation

14.42.4.1 brush() `const QBrush & QwtIntervalSymbol::brush ( ) const`

Returns

Brush

See also

setBrush()

Definition at line 146 of file qwt_interval_symbol.cpp.

14.42.4.2 draw() `void QwtIntervalSymbol::draw ( QPainter * painter, Qt::Orientation orientation, const QPointF & from, const QPointF & to ) const [virtual]`

Draw a symbol depending on its style

Parameters

painter	Painter
orientation	Orientation
from	Start point of the interval in target device coordinates
to	End point of the interval in target device coordinates

See also

setStyle()

Definition at line 200 of file qwt_interval_symbol.cpp.

14.42.4.3 pen() `const QPen & QwtIntervalSymbol::pen ( ) const`

Returns

Pen

See also

setPen(), brush()

Definition at line 185 of file qwt_interval_symbol.cpp.

14.42.4.4 setBrush() `void QwtIntervalSymbol::setBrush ( const QBrush & brush )`

Assign a brush.

The brush is used for the Box style.

Parameters

brush	Brush

See also

brush()

Definition at line 137 of file qwt_interval_symbol.cpp.

14.42.4.5 setPen() [1/2] void QwtIntervalSymbol::setPen (const QColor & *color*, qreal *width* = *0.0*, Qt::PenStyle *style* = *Qt::SolidLine*)

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 164 of file qwt_interval_symbol.cpp.

14.42.4.6 setPen() [2/2] void QwtIntervalSymbol::setPen (const QPen & *pen*)

Assign a pen

Parameters

pen	Pen

See also

pen(), setBrush()

Definition at line 176 of file qwt_interval_symbol.cpp.

14.42.4.7 setStyle() void QwtIntervalSymbol::setStyle (Style *style*)

Specify the symbol style

Parameters

style	Style

See also

style(), Style

Definition at line 94 of file qwt_interval_symbol.cpp.

14.42.4.8 setWidth() `void QwtIntervalSymbol::setWidth ( int width )`

Specify the width of the symbol It is used depending on the style.

Parameters

width	Width

See also

width(), setStyle()

Definition at line 115 of file qwt_interval_symbol.cpp.

14.42.4.9 style() `QwtIntervalSymbol::Style QwtIntervalSymbol::style ( ) const`

Returns

Current symbol style

See also

setStyle()

Definition at line 103 of file qwt_interval_symbol.cpp.

14.42.4.10 width() `int QwtIntervalSymbol::width ( ) const`

Returns

Width of the symbol.

See also

setWidth(), setStyle()

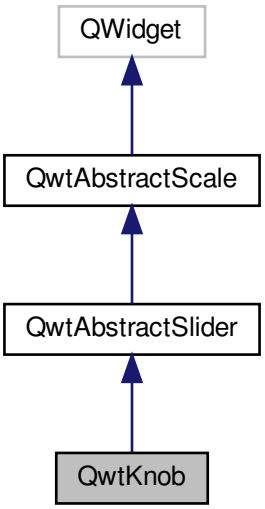
Definition at line 124 of file qwt_interval_symbol.cpp.

14.43 QwtKnob Class Reference

The Knob Widget.

`#include <qwt_knob.h>`

Inheritance diagram for QwtKnob:



Public Types

- enum KnobStyle { Flat , Raised , Sunken , Styled }

  Style of the knob surface.
- enum MarkerStyle {
  NoMarker = -1 , Tick , Triangle , Dot ,
  Nub , Notch }

  Marker type.

Public Member Functions

- QwtKnob (QWidget *parent=NULL)

  Constructor.
- virtual ~QwtKnob ()

  Destructor.
- void setAlignment (Qt::Alignment)

  Set the alignment of the knob.
- Qt::Alignment alignment () const
- void setKnobWidth (int)

  Change the knob's width.
- int knobWidth () const

Return the width of the knob.

- void setNumTurns (int)

  Set the number of turns.
- int numTurns () const
- void setTotalAngle (double angle)

  Set the total angle by which the knob can be turned.
- double totalAngle () const
- void setKnobStyle (KnobStyle)

  Set the knob type.
- KnobStyle knobStyle () const
- void setBorderWidth (int)

  Set the knob's border width.
- int borderWidth () const

  Return the border width.
- void setMarkerStyle (MarkerStyle)

  Set the marker type of the knob.
- MarkerStyle markerStyle () const
- void setMarkerSize (int)

  Set the size of the marker.
- int markerSize () const
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override
- void setScaleDraw (QwtRoundScaleDraw ∗)
- const QwtRoundScaleDraw ∗ scaleDraw () const
- QwtRoundScaleDraw ∗ scaleDraw ()
- QRect knobRect () const

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗) override
- virtual void changeEvent (QEvent ∗) override
- virtual void drawKnob (QPainter ∗, const QRectF &) const

  Draw the knob.
- virtual void drawFocusIndicator (QPainter ∗) const
- virtual void drawMarker (QPainter ∗, const QRectF &, double angle) const

  Draw the marker at the knob's front.
- virtual double scrolledTo (const QPoint &) const override

  Determine the value for a new position of the mouse.
- virtual bool isScrollPosition (const QPoint &) const override

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

Additional Inherited Members

14.43.1 Detailed Description

The Knob Widget.

The QwtKnob widget imitates look and behavior of a volume knob on a radio. It looks similar to QDial - not to QwtDial.

The value range of a knob might be divided into several turns.

The layout of the knob depends on the knobWidth().

- width $>$ 0 The diameter of the knob is fixed and the knob is aligned according to the alignment() flags inside of the contentsRect().
- width $<=$ 0 The knob is extended to the minimum of width/height of the contentsRect() and aligned in the other direction according to alignment().

Setting a fixed knobWidth() is helpful to align several knobs with different scale labels.

Definition at line 42 of file qwt_knob.h.

14.43.2 Member Enumeration Documentation

14.43.2.1 KnobStyle `enum QwtKnob::KnobStyle`

Style of the knob surface.

Depending on the KnobStyle the surface of the knob is filled from the brushes of the widget palette().

See also

setKnobStyle(), knobStyle()

Enumerator

Flat	Fill the knob with a brush from QPalette::Button.
Raised	Build a gradient from QPalette::Midlight and QPalette::Button.
Sunken	Build a gradient from QPalette::Midlight, QPalette::Button and QPalette::Midlight
Styled	Build a radial gradient from QPalette::Button like it is used for QDial in various Qt styles.

Definition at line 66 of file qwt_knob.h.

14.43.2.2 MarkerStyle `enum QwtKnob::MarkerStyle`

Marker type.

The marker indicates the current value on the knob The default setting is a Notch marker.

See also

setMarkerStyle(), setMarkerSize()

Enumerator

NoMarker	Don't paint any marker.
Tick	Paint a single tick in QPalette::ButtonText color.
Triangle	Paint a triangle in QPalette::ButtonText color.
Dot	Paint a circle in QPalette::ButtonText color.
Nub	Draw a raised ellipse with a gradient build from QPalette::Light and QPalette::Mid
Notch	Draw a sunken ellipse with a gradient build from QPalette::Light and QPalette::Mid

Definition at line 95 of file qwt_knob.h.

14.43.3 Constructor & Destructor Documentation

14.43.3.1 QwtKnob() `QwtKnob::QwtKnob ( QWidget * parent = NULL ) [explicit]`

Constructor.

Construct a knob with an angle of 270°. The style is QwtKnob::Raised and the marker style is QwtKnob::Notch. The width of the knob is set to 50 pixels.

Parameters

parent	Parent widget

See also

setTotalAngle()

Definition at line 103 of file qwt_knob.cpp.

14.43.4 Member Function Documentation

14.43.4.1 alignment() `Qt::Alignment QwtKnob::alignment ( ) const`

Returns

Alignment of the knob inside of contentsRect()

See also

setAlignment(), knobWidth(), knobRect()

Definition at line 743 of file qwt_knob.cpp.

14.43.4.2 changeEvent() `void QwtKnob::changeEvent ( QEvent * event ) [override], [protected], [virtual]`

Handle QEvent::StyleChange and QEvent::FontChange;

Parameters

event	Change event

Reimplemented from QwtAbstractScale.

Definition at line 431 of file qwt_knob.cpp.

14.43.4.3 drawFocusIndicator()

```
void QwtKnob::drawFocusIndicator (
    QPainter * painter ) const  [protected], [virtual]
```

Draw the focus indicator

Parameters

painter	Painter

Definition at line 697 of file qwt_knob.cpp.

14.43.4.4 drawKnob()

```
void QwtKnob::drawKnob (
    QPainter * painter,
    const QRectF & knobRect ) const  [protected], [virtual]
```

Draw the knob.

Parameters

painter	painter
knobRect	Bounding rectangle of the knob (without scale)

Definition at line 489 of file qwt_knob.cpp.

14.43.4.5 drawMarker()

```
void QwtKnob::drawMarker (
    QPainter * painter,
    const QRectF & rect,
    double angle ) const  [protected], [virtual]
```

Draw the marker at the knob's front.

Parameters

painter	Painter
rect	Bounding rectangle of the knob without scale
angle	Angle of the marker in degrees (clockwise, 0 at the 12 o'clock position)

Definition at line 576 of file qwt_knob.cpp.

14.43.4.6 isScrollPosition() `bool QwtKnob::isScrollPosition ( const QPoint & pos ) const [override], [protected], [virtual]`

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

Parameters

pos	Mouse position

Return values

True,when	pos is inside the circle of the knob.

See also

scrolledTo()

Implements QwtAbstractSlider.

Definition at line 347 of file qwt_knob.cpp.

14.43.4.7 knobRect() `QRect QwtKnob::knobRect ( ) const`

Calculate the bounding rectangle of the knob without the scale

Returns

Bounding rectangle of the knob

See also

knobWidth(), alignment(), QWidget::contentsRect()

Definition at line 292 of file qwt_knob.cpp.

14.43.4.8 knobStyle() `QwtKnob::KnobStyle QwtKnob::knobStyle ( ) const`

Returns

Marker type of the knob

See also

setKnobStyle(), setBorderWidth()

Definition at line 144 of file qwt_knob.cpp.

14.43.4.9 markerSize() `int QwtKnob::markerSize ( ) const`

Returns

Marker size

See also

setMarkerSize()

Definition at line 825 of file qwt_knob.cpp.

14.43.4.10 markerStyle() `QwtKnob::MarkerStyle QwtKnob::markerStyle ( ) const`

Returns

Marker type of the knob

See also

setMarkerStyle(), setMarkerSize()

Definition at line 168 of file qwt_knob.cpp.

14.43.4.11 minimumSizeHint() `QSize QwtKnob::minimumSizeHint ( ) const [override], [virtual]`

Returns

Minimum size hint

See also

sizeHint()

Definition at line 843 of file qwt_knob.cpp.

14.43.4.12 numTurns() `int QwtKnob::numTurns ( ) const`

Returns

Number of turns.

When the total angle is below 360° numTurns() is ceiled to 1.

See also

setNumTurns(), setTotalAngle(), totalAngle()

Definition at line 245 of file qwt_knob.cpp.

14.43.4.13 paintEvent() `void QwtKnob::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Repaint the knob

Parameters

event	Paint event

Definition at line 451 of file qwt_knob.cpp.

14.43.4.14 scaleDraw() [1/2] QwtRoundScaleDraw * QwtKnob::scaleDraw ( )

Returns

the scale draw of the knob

See also

setScaleDraw()

Definition at line 281 of file qwt_knob.cpp.

14.43.4.15 scaleDraw() [2/2] const QwtRoundScaleDraw * QwtKnob::scaleDraw ( ) const

Returns

the scale draw of the knob

See also

setScaleDraw()

Definition at line 272 of file qwt_knob.cpp.

14.43.4.16 scrolledTo() double QwtKnob::scrolledTo (const QPoint & *pos*) const [override], [protected], [virtual]

Determine the value for a new position of the mouse.

Parameters

pos	Mouse position

Returns

Value for the mouse position

See also

isScrollPosition()

Implements QwtAbstractSlider.

Definition at line 373 of file qwt_knob.cpp.

14.43.4.17 setAlignment() `void QwtKnob::setAlignment ( Qt::Alignment alignment )`

Set the alignment of the knob.

Similar to a QLabel::alignment() the flags decide how to align the knob inside of contentsRect().

The default setting is Qt::AlignCenter

Parameters

alignment	Or'd alignment flags

See also

alignment(), setKnobWidth(), knobRect()

Definition at line 730 of file qwt_knob.cpp.

14.43.4.18 setBorderWidth() `void QwtKnob::setBorderWidth ( int borderWidth )`

Set the knob's border width.

Parameters

borderWidth	new border width

Definition at line 790 of file qwt_knob.cpp.

14.43.4.19 setKnobStyle() `void QwtKnob::setKnobStyle ( KnobStyle knobStyle )`

Set the knob type.

Parameters

knobStyle	Knob type

See also

knobStyle(), setBorderWidth()

Definition at line 131 of file qwt_knob.cpp.

14.43.4.20 setKnobWidth() `void QwtKnob::setKnobWidth ( int width )`

Change the knob's width.

Setting a fixed value for the diameter of the knob is helpful for aligning several knobs in a row.

Parameters

width	New width

See also

knobWidth(), setAlignment()

Note

Modifies the sizePolicy()

Definition at line 759 of file qwt_knob.cpp.

14.43.4.21 setMarkerSize() `void QwtKnob::setMarkerSize ( int size )`

Set the size of the marker.

When setting a size <= 0 the marker will automatically scaled to 40% of the radius of the knob.

See also

markerSize(), markerStyle()

Definition at line 812 of file qwt_knob.cpp.

14.43.4.22 setMarkerStyle() `void QwtKnob::setMarkerStyle ( MarkerStyle markerStyle )`

Set the marker type of the knob.

Parameters

markerStyle	Marker type

See also

markerStyle(), setMarkerSize()

Definition at line 155 of file qwt_knob.cpp.

14.43.4.23 setNumTurns() `void QwtKnob::setNumTurns ( int numTurns )`

Set the number of turns.

When numTurns $>$ 1 the knob can be turned several times around its axis

- otherwise the total angle is floored to 360°.

See also

numTurns(), totalAngle(), setTotalAngle()

Definition at line 219 of file qwt_knob.cpp.

14.43.4.24 setScaleDraw() `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()

Definition at line 259 of file qwt_knob.cpp.

14.43.4.25 setTotalAngle() `void QwtKnob::setTotalAngle ( double angle )`

Set the total angle by which the knob can be turned.

Parameters

angle	Angle in degrees.

The angle has to be between [10, 360] degrees. Angles above 360 (so that the knob can be turned several times around its axis) have to be set using setNumTurns().

The default angle is 270 degrees.

See also

totalAngle(), setNumTurns()

Definition at line 185 of file qwt_knob.cpp.

14.43.4.26 sizeHint() `QSize QwtKnob::sizeHint ( ) const [override], [virtual]`

Returns

sizeHint()

Definition at line 833 of file qwt_knob.cpp.

14.43.4.27 totalAngle() `double QwtKnob::totalAngle ( ) const`

Returns

the total angle

See also

setTotalAngle(), setNumTurns(), numTurns()

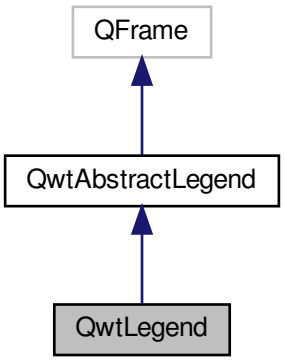
Definition at line 205 of file qwt_knob.cpp.

14.44 QwtLegend Class Reference

The legend widget.

`#include <qwt_legend.h>`

Inheritance diagram for QwtLegend:



Public Slots

- virtual void updateLegend (const QVariant &, const QList< QwtLegendData > &) override

  Update the entries for an item.

Signals

- void clicked (const QVariant &itemInfo, int index)
- void checked (const QVariant &itemInfo, bool on, int index)

Public Member Functions

- QwtLegend (QWidget *parent=NULL)
- virtual ~QwtLegend ()

  Destructor.
- void setMaxColumns (uint numColums)

  Set the maximum number of entries in a row.
- uint maxColumns () const
- void setDefaultItemMode (QwtLegendData::Mode)

  Set the default mode for legend labels.
- QwtLegendData::Mode defaultItemMode () const
- QWidget * contentsWidget ()
- const QWidget * contentsWidget () const
- QWidget * legendWidget (const QVariant &) const
- QList< QWidget * > legendWidgets (const QVariant &) const

- QVariant itemInfo (const QWidget ∗) const
- virtual bool eventFilter (QObject ∗, QEvent ∗) override
- virtual QSize sizeHint () const override

  Return a size hint.
- virtual int heightForWidth (int w) const override
- QScrollBar ∗ horizontalScrollBar () const
- QScrollBar ∗ verticalScrollBar () const
- virtual void renderLegend (QPainter ∗, const QRectF &, bool fillBackground) const override
- virtual void renderItem (QPainter ∗, const QWidget ∗, const QRectF &, bool fillBackground) const
- virtual bool isEmpty () const override
- virtual int scrollExtent (Qt::Orientation) const override

Protected Slots

- void itemClicked ()
- void itemChecked (bool)

Protected Member Functions

- virtual QWidget ∗ createWidget (const QwtLegendData &) const

  Create a widget to be inserted into the legend.
- virtual void updateWidget (QWidget ∗, const QwtLegendData &)

  Update the widget.

14.44.1 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 QwtLegendLabel.

See also

QwtLegendLabel, QwtPlotItem, QwtPlot

Definition at line 31 of file qwt_legend.h.

14.44.2 Constructor & Destructor Documentation

14.44.2.1 QwtLegend() `QwtLegend::QwtLegend ( QWidget * parent = NULL ) [explicit]`

Constructor

Parameters

parent	Parent widget

Definition at line 258 of file qwt_legend.cpp.

14.44.3 Member Function Documentation

14.44.3.1 checked

```
void QwtLegend::checked (
    const QVariant & itemInfo,
    bool on,
    int index )  [signal]
```

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

Parameters

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

Note

clicks are disabled as default

See also

setDefaultItemMode(), defaultItemMode(), QwtPlot::itemToInfo()

14.44.3.2 clicked

```
void QwtLegend::clicked (
    const QVariant & itemInfo,
    int index )  [signal]
```

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

Parameters

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

Note

clicks are disabled as default

See also

setDefaultItemMode(), defaultItemMode(), QwtPlot::itemToInfo()

14.44.3.3 contentsWidget() [1/2] QWidget * QwtLegend::contentsWidget ( )

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

Returns

Container widget of the legend items

Definition at line 355 of file qwt_legend.cpp.

14.44.3.4 contentsWidget() [2/2] const QWidget * QwtLegend::contentsWidget ( ) const

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

Returns

Container widget of the legend items

Definition at line 385 of file qwt_legend.cpp.

14.44.3.5 createWidget() QWidget * QwtLegend::createWidget (const QwtLegendData & *legendData*) const [protected], [virtual]

Create a widget to be inserted into the legend.

The default implementation returns a QwtLegendLabel.

Parameters

legendData	Attributes of the legend entry

Returns

Widget representing data on the legend

Note

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

Definition at line 467 of file qwt_legend.cpp.

14.44.3.6 defaultItemMode() `QwtLegendData::Mode QwtLegend::defaultItemMode ( ) const`

Returns

Default item mode

See also

setDefaultItemMode()

Definition at line 344 of file qwt_legend.cpp.

14.44.3.7 eventFilter() `bool QwtLegend::eventFilter ( QObject * object, QEvent * event ) [override], [virtual]`

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

Parameters

object	Object to be filtered
event	Event

Returns

Forwarded to QwtAbstractLegend::eventFilter()

Definition at line 559 of file qwt_legend.cpp.

14.44.3.8 heightForWidth() `int QwtLegend::heightForWidth ( int width ) const [override], [virtual]`

Returns

The preferred height, for a width.

Parameters

width	Width

Definition at line 538 of file qwt_legend.cpp.

14.44.3.9 horizontalScrollBar() `QScrollBar * QwtLegend::horizontalScrollBar ( ) const`

Returns

Horizontal scrollbar

See also

verticalScrollBar()

Definition at line 364 of file qwt_legend.cpp.

14.44.3.10 isEmpty() `bool QwtLegend::isEmpty ( ) const [override], [virtual]`

Returns

True, when no item is inserted

Implements QwtAbstractLegend.

Definition at line 810 of file qwt_legend.cpp.

14.44.3.11 itemChecked `void QwtLegend::itemChecked ( bool on ) [protected], [slot]`

Called internally when the legend has been checked Emits a checked() signal.

Definition at line 638 of file qwt_legend.cpp.

14.44.3.12 itemClicked `void QwtLegend::itemClicked ( ) [protected], [slot]`

Called internally when the legend has been clicked on. Emits a clicked() signal.

Definition at line 616 of file qwt_legend.cpp.

14.44.3.13 itemInfo() `QVariant QwtLegend::itemInfo ( const QWidget * widget ) const`

Find the item that is associated to a widget

Parameters

widget	Widget on the legend

Returns

Associated item info

See also

legendWidget()

Definition at line 804 of file qwt_legend.cpp.

14.44.3.14 legendWidget() `QWidget * QwtLegend::legendWidget ( const QVariant & itemInfo ) const`

Returns

First widget in the list of widgets associated to an item

Parameters

itemInfo	Info about an item

See also

itemInfo(), QwtPlot::itemToInfo()

Note

Almost all types of items have only one widget

Definition at line 788 of file qwt_legend.cpp.

14.44.3.15 legendWidgets() `QList< QWidget * > QwtLegend::legendWidgets ( const QVariant & itemInfo ) const`

Returns

List of widgets associated to a item

Parameters

itemInfo	Info about an item

See also

legendWidget(), itemInfo(), QwtPlot::itemToInfo()

Definition at line 777 of file qwt_legend.cpp.

14.44.3.16 maxColumns() `uint QwtLegend::maxColumns ( ) const`

Returns

Maximum number of entries in a row

See also

setMaxColumns(), QwtDynGridLayout::maxColumns()

Definition at line 310 of file qwt_legend.cpp.

14.44.3.17 renderItem()

```
void QwtLegend::renderItem (
        QPainter * painter,
        const QWidget * widget,
        const QRectF & rect,
        bool fillBackground ) const  [virtual]
```

Render a legend entry into a given rectangle.

Parameters

painter	Painter
widget	Widget representing a legend entry
rect	Bounding rectangle
fillBackground	When true, fill rect with the widget background

Note

When widget is not derived from QwtLegendLabel renderItem does nothing beside the background

Definition at line 727 of file qwt_legend.cpp.

14.44.3.18 renderLegend() `void QwtLegend::renderLegend ( QPainter * painter, const QRectF & rect, bool fillBackground ) const [override], [virtual]`

Render the legend into a given rectangle.

Parameters

painter	Painter
rect	Bounding rectangle
fillBackground	When true, fill rect with the widget background

See also

renderLegend() is used by QwtPlotRenderer - not by QwtLegend itself

Implements QwtAbstractLegend.

Definition at line 665 of file qwt_legend.cpp.

14.44.3.19 scrollExtent() `int QwtLegend::scrollExtent ( Qt::Orientation orientation ) const [override], [virtual]`

Return the extent, that is needed for the scrollbars

Parameters

orientation	Orientation

Returns

The width of the vertical scrollbar for Qt::Horizontal and v.v.

Reimplemented from QwtAbstractLegend.

Definition at line 821 of file qwt_legend.cpp.

14.44.3.20 setDefaultItemMode() `void QwtLegend::setDefaultItemMode ( QwtLegendData::Mode mode )`

Set the default mode for legend labels.

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

Parameters

mode	Default item mode

See also

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

Note

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

Definition at line 335 of file qwt_legend.cpp.

14.44.3.21 setMaxColumns() `void QwtLegend::setMaxColumns ( uint numColums )`

Set the maximum number of entries in a row.

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

Parameters

numColums	Maximum number of entries in a row

See also

maxColumns(), QwtDynGridLayout::setMaxColumns()

Definition at line 296 of file qwt_legend.cpp.

14.44.3.22 updateLegend `void QwtLegend::updateLegend ( const QVariant & itemInfo, const QList< QwtLegendData > & legendData ) [override], [virtual], [slot]`

Update the entries for an item.

Parameters

itemInfo	Info for an item
legendData	List of legend entry attributes for the item

Definition at line 396 of file qwt_legend.cpp.

14.44.3.23 updateWidget()

```
void QwtLegend::updateWidget (
    QWidget * widget,
    const QwtLegendData & legendData )  [protected], [virtual]
```

Update the widget.

Parameters

widget	Usually a QwtLegendLabel
legendData	Attributes to be displayed

See also

createWidget()

Note

When widget is no QwtLegendLabel updateWidget() does nothing.

Definition at line 489 of file qwt_legend.cpp.

14.44.3.24 verticalScrollBar()

```
QScrollBar * QwtLegend::verticalScrollBar ( ) const
```

Returns

Vertical scrollbar

See also

horizontalScrollBar()

Definition at line 373 of file qwt_legend.cpp.

14.45 QwtLegendData Class Reference

Attributes of an entry on a legend.

```
#include <qwt_legend_data.h>
```

Public Types

- enum Mode { ReadOnly , Clickable , Checkable }

  Mode defining how a legend entry interacts.
- enum Role { **ModeRole** , **TitleRole** , **IconRole** , **UserRole** = 32 }

  Identifier how to interpret a QVariant.

Public Member Functions

- QwtLegendData ()

  Constructor.
- ~QwtLegendData ()

  Destructor.
- void setValues (const QMap< int, QVariant > &)
- const QMap< int, QVariant > & values () const
- void setValue (int role, const QVariant &)
- QVariant value (int role) const
- bool hasRole (int role) const
- bool isValid () const
- QwtGraphic icon () const
- QwtText title () const
- Mode mode () const

14.45.1 Detailed Description

Attributes of an entry on a legend.

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

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

See also

QwtLegend, QwtPlotLegendItem

Note

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

Definition at line 36 of file qwt_legend_data.h.

14.45.2 Member Enumeration Documentation

14.45.2.1 Mode `enum QwtLegendData::Mode`

Mode defining how a legend entry interacts.

Enumerator

ReadOnly	The legend item is not interactive, like a label.
Clickable	The legend item is clickable, like a push button.
Checkable	The legend item is checkable, like a checkable button.

Definition at line 40 of file qwt_legend_data.h.

14.45.3 Member Function Documentation

14.45.3.1 hasRole() `bool QwtLegendData::hasRole ( int role ) const`

Parameters

role	Attribute role

Returns

True, when the internal map has an entry for role

Definition at line 51 of file qwt_legend_data.cpp.

14.45.3.2 icon() `QwtGraphic QwtLegendData::icon ( ) const`

Returns

Value of the IconRole attribute

Definition at line 106 of file qwt_legend_data.cpp.

14.45.3.3 isValid() `bool QwtLegendData::isValid ( ) const`

Returns

True, when the internal map is empty

Definition at line 82 of file qwt_legend_data.cpp.

14.45.3.4 mode() `QwtLegendData::Mode QwtLegendData::mode ( ) const`

Returns

Value of the ModeRole attribute

Definition at line 120 of file qwt_legend_data.cpp.

14.45.3.5 setValue() `void QwtLegendData::setValue ( int role, const QVariant & data )`

Set an attribute value

Parameters

role	Attribute role
data	Attribute value

See also

value()

Definition at line 64 of file qwt_legend_data.cpp.

14.45.3.6 setValues() `void QwtLegendData::setValues ( const QMap< int, QVariant > & map )`

Set the legend attributes

QwtLegendData actually is a QMap<int, QVariant> with some convenience interfaces

Parameters

map	Values

See also

values()

Definition at line 33 of file qwt_legend_data.cpp.

14.45.3.7 title() `QwtText QwtLegendData::title ( ) const`

Returns

Value of the TitleRole attribute

Definition at line 88 of file qwt_legend_data.cpp.

14.45.3.8 value() `QVariant QwtLegendData::value ( int role ) const`

Parameters

role	Attribute role

Returns

Attribute value for a specific role

Definition at line 73 of file qwt_legend_data.cpp.

14.45.3.9 values() `const QMap< int, QVariant > & QwtLegendData::values ( ) const`

Returns

Legend attributes

See also

setValues()

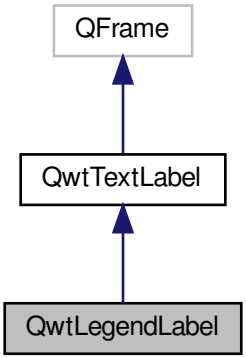
Definition at line 42 of file qwt_legend_data.cpp.

14.46 QwtLegendLabel Class Reference

A widget representing something on a QwtLegend.

`#include <qwt_legend_label.h>`

Inheritance diagram for QwtLegendLabel:



Public Slots

- void setChecked (bool on)

Signals

- void clicked ()

  Signal, when the legend item has been clicked.
- void pressed ()

  Signal, when the legend item has been pressed.
- void released ()

  Signal, when the legend item has been released.
- void checked (bool)

  Signal, when the legend item has been toggled.

Public Member Functions

- QwtLegendLabel (QWidget *parent=0)
- virtual ~QwtLegendLabel ()

  Destructor.
- void setData (const QwtLegendData &)
- const QwtLegendData & data () const
- void setItemMode (QwtLegendData::Mode)
- QwtLegendData::Mode itemMode () const
- void setSpacing (int spacing)

  Change the spacing between icon and text.
- int spacing () const
- virtual void setText (const QwtText &) override
- void setIcon (const QPixmap &)
- QPixmap icon () const
- virtual QSize sizeHint () const override

  Return a size hint.
- bool isChecked () const

  Return true, if the item is checked.

Protected Member Functions

- void setDown (bool)

  Set the item being down.
- bool isDown () const

  Return true, if the item is down.
- virtual void paintEvent (QPaintEvent *) override

  Paint event.
- virtual void mousePressEvent (QMouseEvent *) override

  Handle mouse press events.
- virtual void mouseReleaseEvent (QMouseEvent *) override

  Handle mouse release events.
- virtual void keyPressEvent (QKeyEvent *) override

  Handle key press events.
- virtual void keyReleaseEvent (QKeyEvent *) override

  Handle key release events.

14.46.1 Detailed Description

A widget representing something on a QwtLegend.

Definition at line 22 of file qwt_legend_label.h.

14.46.2 Constructor & Destructor Documentation

14.46.2.1 QwtLegendLabel() `QwtLegendLabel::QwtLegendLabel ( QWidget * parent = 0 ) [explicit]`

Parameters

parent	Parent widget

Definition at line 91 of file qwt_legend_label.cpp.

14.46.3 Member Function Documentation

14.46.3.1 data() `const QwtLegendData & QwtLegendLabel::data ( ) const`

Returns

Attributes of the label

See also

setData(), QwtPlotItem::legendData()

Definition at line 83 of file qwt_legend_label.cpp.

14.46.3.2 icon() `QPixmap QwtLegendLabel::icon ( ) const`

Returns

Pixmap representing a plot item

See also

setIcon()

Definition at line 176 of file qwt_legend_label.cpp.

14.46.3.3 itemMode() `QwtLegendData::Mode QwtLegendLabel::itemMode ( ) const`

Returns

Item mode

See also

setItemMode()

Definition at line 149 of file qwt_legend_label.cpp.

14.46.3.4 setChecked `void QwtLegendLabel::setChecked ( bool on ) [slot]`

Check/Uncheck a the item

Parameters

on	check/uncheck

See also

setItemMode()

Definition at line 217 of file qwt_legend_label.cpp.

14.46.3.5 setData() `void QwtLegendLabel::setData ( const QwtLegendData & legendData )`

Set the attributes of the legend label

Parameters

legendData	Attributes of the label

See also

data()

Definition at line 61 of file qwt_legend_label.cpp.

14.46.3.6 setIcon() void QwtLegendLabel::setIcon (const QPixmap & *icon*)

Assign the icon

Parameters

icon	Pixmap representing a plot item

See also

icon(), QwtPlotItem::legendIcon()

Definition at line 161 of file qwt_legend_label.cpp.

14.46.3.7 setItemMode() `void QwtLegendLabel::setItemMode ( QwtLegendData::Mode mode )`

Set the item mode The default is QwtLegendData::ReadOnly

Parameters

mode	Item mode

See also

itemMode()

Definition at line 130 of file qwt_legend_label.cpp.

14.46.3.8 setSpacing() `void QwtLegendLabel::setSpacing ( int spacing )`

Change the spacing between icon and text.

Parameters

spacing	Spacing

See also

spacing(), QwtTextLabel::margin()

Definition at line 187 of file qwt_legend_label.cpp.

14.46.3.9 setText() `void QwtLegendLabel::setText ( const QwtText & text ) [override], [virtual]`

Set the text to the legend item

Parameters

text	Text label

See also

QwtTextLabel::text()

Reimplemented from QwtTextLabel.

Definition at line 112 of file qwt_legend_label.cpp.

14.46.3.10 spacing() `int QwtLegendLabel::spacing ( ) const`

Returns

Spacing between icon and text

See also

setSpacing(), QwtTextLabel::margin()

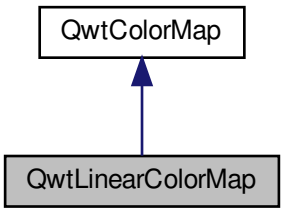
Definition at line 206 of file qwt_legend_label.cpp.

14.47 QwtLinearColorMap Class Reference

QwtLinearColorMap builds a color map from color stops.

`#include <qwt_color_map.h>`

Inheritance diagram for QwtLinearColorMap:



Public Types

- enum Mode { FixedColors , ScaledColors }

Public Member Functions

- QwtLinearColorMap (QwtColorMap::Format=QwtColorMap::RGB)
- QwtLinearColorMap (const QColor &from, const QColor &to, QwtColorMap::Format=QwtColorMap::RGB)
- virtual ~QwtLinearColorMap ()

  Destructor.
- void setMode (Mode)

  Set the mode of the color map.
- Mode mode () const
- void setColorInterval (const QColor &color1, const QColor &color2)
- void addColorStop (double value, const QColor &)
- QVector< double > colorStops () const
- QColor color1 () const
- QColor color2 () const
- virtual QRgb rgb (const QwtInterval &, double value) const override
- virtual uint colorIndex (int numColors, const QwtInterval &, double value) const override

  Map a value of a given interval into a color index.

Additional Inherited Members

14.47.1 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 according to mode().

Definition at line 98 of file qwt_color_map.h.

14.47.2 Member Enumeration Documentation

14.47.2.1 Mode enum QwtLinearColorMap::Mode

Mode of color map

See also

setMode(), mode()

Enumerator

FixedColors	Return the color from the next lower color stop.
ScaledColors	Interpolating the colors of the adjacent stops.

Definition at line 105 of file qwt_color_map.h.

14.47.3 Constructor & Destructor Documentation

14.47.3.1 QwtLinearColorMap() [1/2] QwtLinearColorMap::QwtLinearColorMap (QwtColorMap::Format format = QwtColorMap::RGB) [explicit]

Build a color map with two stops at 0.0 and 1.0. The color at 0.0 is Qt::blue, at 1.0 it is Qt::yellow.

Parameters

format	Preferred format of the color map

Definition at line 351 of file qwt_color_map.cpp.

14.47.3.2 QwtLinearColorMap() [2/2] 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

color1	Color used for the minimum value of the value interval
color2	Color used for the maximum value of the value interval
format	Preferred format for the color map

Definition at line 367 of file qwt_color_map.cpp.

14.47.4 Member Function Documentation

14.47.4.1 addColorStop() 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

value	Value between [0.0, 1.0]
color	Color stop

Definition at line 433 of file qwt_color_map.cpp.

14.47.4.2 color1() `QColor QwtLinearColorMap::color1 ( ) const`

Returns

the first color of the color range

See also

setColorInterval()

Definition at line 451 of file qwt_color_map.cpp.

14.47.4.3 color2() `QColor QwtLinearColorMap::color2 ( ) const`

Returns

the second color of the color range

See also

setColorInterval()

Definition at line 460 of file qwt_color_map.cpp.

14.47.4.4 colorIndex()

```
uint QwtLinearColorMap::colorIndex (
        int numColors,
        const QwtInterval & interval,
        double value ) const  [override], [virtual]
```

Map a value of a given interval into a color index.

Parameters

numColors	Size of the color table
interval	Range for all values
value	Value to map into a color index

Returns

Index, between 0 and 255

Note

NaN values are mapped to 0

Reimplemented from QwtColorMap.

Definition at line 494 of file qwt_color_map.cpp.

14.47.4.5 colorStops() `QVector< double > QwtLinearColorMap::colorStops ( ) const`

Returns

Positions of color stops in increasing order

Definition at line 442 of file qwt_color_map.cpp.

14.47.4.6 mode() `QwtLinearColorMap::Mode QwtLinearColorMap::mode ( ) const`

Returns

Mode of the color map

See also

setMode()

Definition at line 400 of file qwt_color_map.cpp.

14.47.4.7 rgb()

```
QRgb QwtLinearColorMap::rgb (
        const QwtInterval & interval,
        double value ) const  [override], [virtual]
```

Map a value of a given interval into a RGB value

Parameters

interval	Range for all values
value	Value to map into a RGB value

Returns

RGB value for value

Implements QwtColorMap.

Definition at line 473 of file qwt_color_map.cpp.

14.47.4.8 setColorInterval() void QwtLinearColorMap::setColorInterval (const QColor & color1, const QColor & color2)

Set the color range

Add stops at 0.0 and 1.0.

Parameters

color1	Color used for the minimum value of the value interval
color2	Color used for the maximum value of the value interval

See also

color1(), color2()

Definition at line 415 of file qwt_color_map.cpp.

14.47.4.9 setMode() 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()

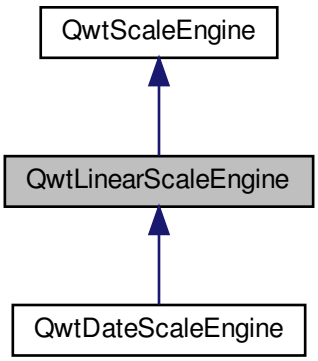
Definition at line 391 of file qwt_color_map.cpp.

14.48 QwtLinearScaleEngine Class Reference

A scale engine for linear scales.

```
#include <qwt_scale_engine.h>
```

Inheritance diagram for QwtLinearScaleEngine:



Public Member Functions

- QwtLinearScaleEngine (uint base=10)
- virtual ~QwtLinearScaleEngine ()

  Destructor.
- virtual void autoScale (int maxNumSteps, double &x1, double &x2, double &stepSize) const override
- virtual QwtScaleDiv divideScale (double x1, double x2, int maxMajorSteps, int maxMinorSteps, double stepSize=0.0) const override

  Calculate a scale division for an interval.

Protected Member Functions

- QwtInterval align (const QwtInterval &, double stepSize) const

  Align an interval to a step size.
- void buildTicks (const QwtInterval &, double stepSize, int maxMinorSteps, QList< double > ticks[QwtScaleDiv::NTickTypes]) const

  Calculate ticks for an interval.
- QList< double > buildMajorTicks (const QwtInterval &interval, double stepSize) const

  Calculate major ticks for an interval.
- void buildMinorTicks (const QList< double > &majorTicks, int maxMinorSteps, double stepSize, QList< double > &minorTicks, QList< double > &mediumTicks) const

  Calculate minor/medium ticks for major ticks.

Additional Inherited Members

14.48.1 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.

Definition at line 151 of file qwt_scale_engine.h.

14.48.2 Constructor & Destructor Documentation

14.48.2.1 QwtLinearScaleEngine() QwtLinearScaleEngine::QwtLinearScaleEngine (uint *base* = *10*) [explicit]

Constructor

Parameters

base	Base of the scale engine

See also

setBase()

Definition at line 511 of file qwt_scale_engine.cpp.

14.48.3 Member Function Documentation

14.48.3.1 align() QwtInterval QwtLinearScaleEngine::align (const QwtInterval & *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

interval	Interval
stepSize	Step size

Returns

Aligned interval

Definition at line 741 of file qwt_scale_engine.cpp.

14.48.3.2 autoScale() void QwtLinearScaleEngine::autoScale (int *maxNumSteps*, double & *x1*,

```
            double & x2,
            double & stepSize ) const  [override], [virtual]
```

Align and divide an interval

Parameters

maxNumSteps	Max. number of steps
x1	First limit of the interval (In/Out)
x2	Second limit of the interval (In/Out)
stepSize	Step size (Out)

See also

setAttribute()

Implements QwtScaleEngine.

Reimplemented in QwtDateScaleEngine.

Definition at line 531 of file qwt_scale_engine.cpp.

14.48.3.3 buildMajorTicks()

```
QList< double > QwtLinearScaleEngine::buildMajorTicks (
    const QwtInterval & interval,
    double stepSize ) const  [protected]
```

Calculate major ticks for an interval.

Parameters

interval	Interval
stepSize	Step size

Returns

Calculated ticks

Definition at line 664 of file qwt_scale_engine.cpp.

14.48.3.4 buildMinorTicks()

```
void QwtLinearScaleEngine::buildMinorTicks (
    const QList< double > & majorTicks,
    int maxMinorSteps,
    double stepSize,
    QList< double > & minorTicks,
    QList< double > & mediumTicks ) const  [protected]
```

Calculate minor/medium ticks for major ticks.

Parameters

majorTicks	Major ticks
maxMinorSteps	Maximum number of minor steps
stepSize	Step size
minorTicks	Array to be filled with the calculated minor ticks
mediumTicks	Array to be filled with the calculated medium ticks

Definition at line 692 of file qwt_scale_engine.cpp.

14.48.3.5 buildTicks()

```
void QwtLinearScaleEngine::buildTicks (
    const QwtInterval & interval,
    double stepSize,
    int maxMinorSteps,
    QList< double > ticks[QwtScaleDiv::NTickTypes] ) const  [protected]
```

Calculate ticks for an interval.

Parameters

interval	Interval
stepSize	Step size
maxMinorSteps	Maximum number of minor steps
ticks	Arrays to be filled with the calculated ticks

See also

buildMajorTicks(), buildMinorTicks

Definition at line 627 of file qwt_scale_engine.cpp.

14.48.3.6 divideScale()

```
QwtScaleDiv QwtLinearScaleEngine::divideScale (
    double x1,
    double x2,
    int maxMajorSteps,
    int maxMinorSteps,
    double stepSize = 0.0 ) const  [override], [virtual]
```

Calculate a scale division for an interval.

Parameters

x1	First interval limit
x2	Second interval limit
maxMajorSteps	Maximum for the number of major steps
maxMinorSteps	Maximum number of minor steps
stepSize	Step size. If stepSize == 0, the engine calculates one.

Returns

Calculated scale division

Implements QwtScaleEngine.

Reimplemented in QwtDateScaleEngine.

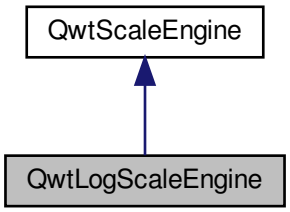
Definition at line 577 of file qwt_scale_engine.cpp.

14.49 QwtLogScaleEngine Class Reference

A scale engine for logarithmic scales.

`#include <qwt_scale_engine.h>`

Inheritance diagram for QwtLogScaleEngine:



Public Member Functions

- QwtLogScaleEngine (uint base=10)
- virtual ~QwtLogScaleEngine ()

  Destructor.
- virtual void autoScale (int maxNumSteps, double &x1, double &x2, double &stepSize) const override
- virtual QwtScaleDiv divideScale (double x1, double x2, int maxMajorSteps, int maxMinorSteps, double stepSize=0.0) const override

  Calculate a scale division for an interval.

Protected Member Functions

- QwtInterval align (const QwtInterval &, double stepSize) const

  Align an interval to a step size.
- void buildTicks (const QwtInterval &, double stepSize, int maxMinorSteps, QList< double > ticks[QwtScaleDiv::NTickTypes]) const

  Calculate ticks for an interval.
- QList< double > buildMajorTicks (const QwtInterval &interval, double stepSize) const

  Calculate major ticks for an interval.
- void buildMinorTicks (const QList< double > &majorTicks, int maxMinorSteps, double stepSize, QList< double > &minorTicks, QList< double > &mediumTicks) const

  Calculate minor/medium ticks for major ticks.

Additional Inherited Members

14.49.1 Detailed Description

A scale engine for logarithmic scales.

The step size is measured in *decades* and the major step size will be adjusted to fit the pattern $\{1, 2, 3, 5\} \cdot 10^n$, where n is a natural number including zero.

Warning

the step size as well as the margins are measured in *decades*.

Definition at line 191 of file qwt_scale_engine.h.

14.49.2 Constructor & Destructor Documentation

14.49.2.1 QwtLogScaleEngine() `QwtLogScaleEngine::QwtLogScaleEngine ( uint base = 10 ) [explicit]`

Constructor

Parameters

base	Base of the scale engine

See also

setBase()

Definition at line 776 of file qwt_scale_engine.cpp.

14.49.3 Member Function Documentation

14.49.3.1 align() `QwtInterval QwtLogScaleEngine::align ( const QwtInterval & 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

interval	Interval
stepSize	Step size

Returns

Aligned interval

Definition at line 1106 of file qwt_scale_engine.cpp.

14.49.3.2 autoScale() `void QwtLogScaleEngine::autoScale ( int maxNumSteps, double & x1,`

```
            double & x2,
            double & stepSize ) const  [override], [virtual]
```

Align and divide an interval

Parameters

maxNumSteps	Max. number of steps
x1	First limit of the interval (In/Out)
x2	Second limit of the interval (In/Out)
stepSize	Step size (Out)

See also

QwtScaleEngine::setAttribute()

Implements QwtScaleEngine.

Definition at line 797 of file qwt_scale_engine.cpp.

14.49.3.3 buildMajorTicks()

```
QList< double > QwtLogScaleEngine::buildMajorTicks (
    const QwtInterval & interval,
    double stepSize ) const  [protected]
```

Calculate major ticks for an interval.

Parameters

interval	Interval
stepSize	Step size

Returns

Calculated ticks

Definition at line 967 of file qwt_scale_engine.cpp.

14.49.3.4 buildMinorTicks()

```
void QwtLogScaleEngine::buildMinorTicks (
    const QList< double > & majorTicks,
    int maxMinorSteps,
    double stepSize,
    QList< double > & minorTicks,
    QList< double > & mediumTicks ) const  [protected]
```

Calculate minor/medium ticks for major ticks.

Parameters

majorTicks	Major ticks
maxMinorSteps	Maximum number of minor steps
stepSize	Step size
minorTicks	Array to be filled with the calculated minor ticks
mediumTicks	Array to be filled with the calculated medium ticks

Definition at line 1002 of file qwt_scale_engine.cpp.

14.49.3.5 buildTicks()

```
void QwtLogScaleEngine::buildTicks (
    const QwtInterval & interval,
    double stepSize,
    int maxMinorSteps,
    QList< double > ticks[QwtScaleDiv::NTickTypes] ) const  [protected]
```

Calculate ticks for an interval.

Parameters

interval	Interval
maxMinorSteps	Maximum number of minor steps
stepSize	Step size
ticks	Arrays to be filled with the calculated ticks

See also

buildMajorTicks(), buildMinorTicks

Definition at line 940 of file qwt_scale_engine.cpp.

14.49.3.6 divideScale()

```
QwtScaleDiv QwtLogScaleEngine::divideScale (
    double x1,
    double x2,
    int maxMajorSteps,
    int maxMinorSteps,
    double stepSize = 0.0 ) const  [override], [virtual]
```

Calculate a scale division for an interval.

Parameters

x1	First interval limit
x2	Second interval limit
maxMajorSteps	Maximum for the number of major steps
maxMinorSteps	Maximum number of minor steps
stepSize	Step size. If stepSize == 0, the engine calculates one.

Returns

Calculated scale division

Implements QwtScaleEngine.

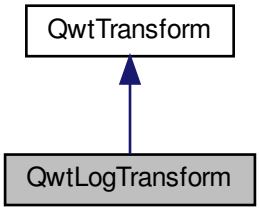
Definition at line 879 of file qwt_scale_engine.cpp.

14.50 QwtLogTransform Class Reference

Logarithmic transformation.

`#include <qwt_transform.h>`

Inheritance diagram for QwtLogTransform:



Public Member Functions

- QwtLogTransform ()
  Constructor.
- virtual ~QwtLogTransform ()
  Destructor.
- virtual double transform (double value) const override
- virtual double invTransform (double value) const override
- virtual double bounded (double value) const override
- virtual QwtTransform * copy () const override

Static Public Attributes

- static const double LogMin = 1.0e-150
  Smallest allowed value for logarithmic scales: 1.0e-150.
- static const double LogMax = 1.0e150
  Largest allowed value for logarithmic scales: 1.0e150.

14.50.1 Detailed Description

Logarithmic transformation.

QwtLogTransform modifies the values using log() and exp().

Note

In the calculations of QwtScaleMap the base of the log function has no effect on the mapping. So QwtLogTransform can be used for log2(), log10() or any other logarithmic scale.

Definition at line 100 of file qwt_transform.h.

14.50.2 Member Function Documentation

14.50.2.1 bounded() `double QwtLogTransform::bounded ( double value ) const [override], [virtual]`

Parameters

value	Value to be bounded

Returns

qBound(LogMin, value, LogMax)

Reimplemented from QwtTransform.

Definition at line 106 of file qwt_transform.cpp.

14.50.2.2 copy() `QwtTransform * QwtLogTransform::copy ( ) const [override], [virtual]`

Returns

Clone of the transformation

Implements QwtTransform.

Definition at line 112 of file qwt_transform.cpp.

14.50.2.3 invTransform() `double QwtLogTransform::invTransform ( double value ) const [override], [virtual]`

Parameters

value	Value to be transformed

Returns

exp(value)

Implements QwtTransform.

Definition at line 97 of file qwt_transform.cpp.

14.50.2.4 transform() double QwtLogTransform::transform (double *value*) const [override], [virtual]

Parameters

value	Value to be transformed

Returns

log(value)

Implements QwtTransform.

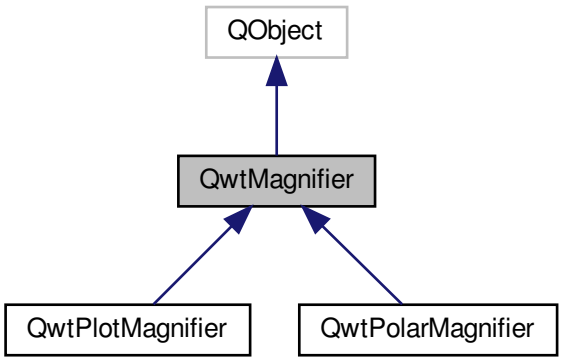
Definition at line 88 of file qwt_transform.cpp.

14.51 QwtMagnifier Class Reference

QwtMagnifier provides zooming, by magnifying in steps.

`#include <qwt_magnifier.h>`

Inheritance diagram for QwtMagnifier:



Public Member Functions

- QwtMagnifier (QWidget ∗)
- virtual ∼QwtMagnifier ()

  Destructor.
- QWidget ∗ parentWidget ()
- const QWidget ∗ parentWidget () const
- void setEnabled (bool)

  En/disable the magnifier.
- bool isEnabled () const

- void setMouseFactor (double)

  Change the mouse factor.
- double mouseFactor () const
- void setMouseButton (Qt::MouseButton, Qt::KeyboardModifiers=Qt::NoModifier)
- void getMouseButton (Qt::MouseButton &, Qt::KeyboardModifiers &) const
- void setWheelFactor (double)

  Change the wheel factor.
- double wheelFactor () const
- void setWheelModifiers (Qt::KeyboardModifiers)
- Qt::KeyboardModifiers wheelModifiers () const
- void setKeyFactor (double)

  Change the key factor.
- double keyFactor () const
- void setZoomInKey (int key, Qt::KeyboardModifiers=Qt::NoModifier)
- void getZoomInKey (int &key, Qt::KeyboardModifiers &) const

  Retrieve the settings of the zoom in key.
- void setZoomOutKey (int key, Qt::KeyboardModifiers=Qt::NoModifier)
- void getZoomOutKey (int &key, Qt::KeyboardModifiers &) const

  Retrieve the settings of the zoom out key.
- virtual bool eventFilter (QObject ∗, QEvent ∗) override

  Event filter.

Protected Member Functions

- virtual void rescale (double factor)=0
- virtual void widgetMousePressEvent (QMouseEvent ∗)
- virtual void widgetMouseReleaseEvent (QMouseEvent ∗)
- virtual void widgetMouseMoveEvent (QMouseEvent ∗)
- virtual void widgetWheelEvent (QWheelEvent ∗)
- virtual void widgetKeyPressEvent (QKeyEvent ∗)
- virtual void widgetKeyReleaseEvent (QKeyEvent ∗)

14.51.1 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.

Definition at line 27 of file qwt_magnifier.h.

14.51.2 Constructor & Destructor Documentation

14.51.2.1 QwtMagnifier() `QwtMagnifier::QwtMagnifier ( QWidget * parent ) [explicit]`

Constructor

Parameters

parent	Widget to be magnified

Definition at line 63 of file qwt_magnifier.cpp.

14.51.3 Member Function Documentation

14.51.3.1 eventFilter()

```
bool QwtMagnifier::eventFilter (
    QObject * object,
    QEvent * event )  [override], [virtual]
```

Event filter.

When isEnabled() is true, the mouse events of the observed widget are filtered.

Parameters

object	Object to be filtered
event	Event

Returns

Forwarded to QObject::eventFilter()

See also

widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent() widgetKeyReleaseEvent()

Definition at line 317 of file qwt_magnifier.cpp.

14.51.3.2 getMouseButton()

```
void QwtMagnifier::getMouseButton (
    Qt::MouseButton & button,
    Qt::KeyboardModifiers & modifiers ) const
```

See also

setMouseButton()

Definition at line 210 of file qwt_magnifier.cpp.

14.51.3.3 getZoomInKey()

```
void QwtMagnifier::getZoomInKey (
    int & key,
    Qt::KeyboardModifiers & modifiers ) const
```

Retrieve the settings of the zoom in key.

Parameters

key	Key code, see Qt::Key
modifiers	Keyboard modifiers

See also

setZoomInKey()

Definition at line 265 of file qwt_magnifier.cpp.

14.51.3.4 getZoomOutKey() `void QwtMagnifier::getZoomOutKey ( int & key, Qt::KeyboardModifiers & modifiers ) const`

Retrieve the settings of the zoom out key.

Parameters

key	Key code, see Qt::Key
modifiers	Keyboard modifiers

See also

setZoomOutKey()

Definition at line 295 of file qwt_magnifier.cpp.

14.51.3.5 isEnabled() `bool QwtMagnifier::isEnabled ( ) const`

Returns

true when enabled, false otherwise

See also

setEnabled(), eventFilter()

Definition at line 113 of file qwt_magnifier.cpp.

14.51.3.6 keyFactor() `double QwtMagnifier::keyFactor ( ) const`

Returns

Key factor

See also

setKeyFactor()

Definition at line 237 of file qwt_magnifier.cpp.

14.51.3.7 mouseFactor() `double QwtMagnifier::mouseFactor ( ) const`

Returns

Mouse factor

See also

setMouseFactor()

Definition at line 188 of file qwt_magnifier.cpp.

14.51.3.8 parentWidget() `[1/2]` `QWidget * QwtMagnifier::parentWidget ( )`

Returns

Parent widget, where the rescaling happens

Definition at line 499 of file qwt_magnifier.cpp.

14.51.3.9 parentWidget() `[2/2]` `const QWidget * QwtMagnifier::parentWidget ( ) const`

Returns

Parent widget, where the rescaling happens

Definition at line 505 of file qwt_magnifier.cpp.

14.51.3.10 rescale() `virtual void QwtMagnifier::rescale ( double factor ) [protected], [pure virtual]`

Rescale the parent widget

Parameters

factor	Scale factor

14.51.3.11 setEnabled() `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

on	true or false

See also

isEnabled(), eventFilter()

Definition at line 92 of file qwt_magnifier.cpp.

14.51.3.12 setKeyFactor() `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

factor	Key factor

See also

keyFactor(), setZoomInKey(), setZoomOutKey(), setWheelFactor, setMouseFactor()

Definition at line 228 of file qwt_magnifier.cpp.

14.51.3.13 setMouseButton() `void QwtMagnifier::setMouseButton ( Qt::MouseButton button, Qt::KeyboardModifiers modifiers = Qt::NoModifier )`

Assign the mouse button, that is used for zooming in/out. The default value is Qt::RightButton.

Parameters

button	Button
modifiers	Keyboard modifiers

See also

getMouseButton()

Definition at line 202 of file qwt_magnifier.cpp.

14.51.3.14 setMouseFactor() `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

factor	Wheel factor

See also

mouseFactor(), setMouseButton(), setWheelFactor(), setKeyFactor()

Definition at line 179 of file qwt_magnifier.cpp.

14.51.3.15 setWheelFactor() `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.

Use values > 1 for magnification (i.e. 2.0) and values < 1 for scaling down (i.e. 1/2.0 = 0.5). You can use this feature for inverting the direction of the wheel.

The default value is 0.9.

Parameters

factor	Wheel factor

See also

wheelFactor(), setWheelButtonState(), setMouseFactor(), setKeyFactor()

Definition at line 134 of file qwt_magnifier.cpp.

14.51.3.16 setWheelModifiers() `void QwtMagnifier::setWheelModifiers ( Qt::KeyboardModifiers modifiers )`

Assign keyboard modifiers for zooming in/out using the wheel. The default modifiers are Qt::NoModifiers.

Parameters

modifiers	Keyboard modifiers

See also

wheelModifiers()

Definition at line 155 of file qwt_magnifier.cpp.

14.51.3.17 setZoomInKey() `void QwtMagnifier::setZoomInKey ( int key, Qt::KeyboardModifiers modifiers = Qt::NoModifier )`

Assign the key, that is used for zooming in. The default combination is Qt::Key_Plus + Qt::NoModifier.

Parameters

key	
modifiers	

See also

getZoomInKey(), setZoomOutKey()

Definition at line 250 of file qwt_magnifier.cpp.

14.51.3.18 setZoomOutKey() `void QwtMagnifier::setZoomOutKey ( int key, Qt::KeyboardModifiers modifiers = Qt::NoModifier )`

Assign the key, that is used for zooming out. The default combination is Qt::Key_Minus + Qt::NoModifier.

Parameters

key	
modifiers	

See also

getZoomOutKey(), setZoomOutKey()

Definition at line 280 of file qwt_magnifier.cpp.

14.51.3.19 wheelFactor() `double QwtMagnifier::wheelFactor ( ) const`

Returns

Wheel factor

See also

setWheelFactor()

Definition at line 143 of file qwt_magnifier.cpp.

14.51.3.20 wheelModifiers() `Qt::KeyboardModifiers QwtMagnifier::wheelModifiers ( ) const`

Returns

Wheel modifiers

See also

setWheelModifiers()

Definition at line 164 of file qwt_magnifier.cpp.

14.51.3.21 widgetKeyPressEvent() `void QwtMagnifier::widgetKeyPressEvent ( QKeyEvent * keyEvent ) [protected], [virtual]`

Handle a key press event for the observed widget.

Parameters

keyEvent	Key event

See also

eventFilter(), widgetKeyReleaseEvent()

Reimplemented in QwtPolarMagnifier.

Definition at line 473 of file qwt_magnifier.cpp.

14.51.3.22 widgetKeyReleaseEvent()

```
void QwtMagnifier::widgetKeyReleaseEvent (
        QKeyEvent * keyEvent )  [protected], [virtual]
```

Handle a key release event for the observed widget.

Parameters

keyEvent	Key event

See also

eventFilter(), widgetKeyReleaseEvent()

Definition at line 493 of file qwt_magnifier.cpp.

14.51.3.23 widgetMouseMoveEvent()

```
void QwtMagnifier::widgetMouseMoveEvent (
        QMouseEvent * mouseEvent )  [protected], [virtual]
```

Handle a mouse move event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(),

Definition at line 407 of file qwt_magnifier.cpp.

14.51.3.24 widgetMousePressEvent() void QwtMagnifier::widgetMousePressEvent (QMouseEvent * *mouseEvent*) [protected], [virtual]

Handle a mouse press event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMouseReleaseEvent(), widgetMouseMoveEvent()

Definition at line 365 of file qwt_magnifier.cpp.

14.51.3.25 widgetMouseReleaseEvent()

```
void QwtMagnifier::widgetMouseReleaseEvent (
        QMouseEvent * mouseEvent ) [protected], [virtual]
```

Handle a mouse release event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseMoveEvent(),

Definition at line 390 of file qwt_magnifier.cpp.

14.51.3.26 widgetWheelEvent()

```
void QwtMagnifier::widgetWheelEvent (
        QWheelEvent * wheelEvent ) [protected], [virtual]
```

Handle a wheel event for the observed widget.

Parameters

wheelEvent	Wheel event

See also

eventFilter()

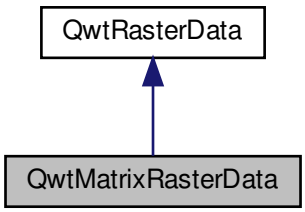
Definition at line 431 of file qwt_magnifier.cpp.

14.52 QwtMatrixRasterData Class Reference

A class representing a matrix of values as raster data.

`#include <qwt_matrix_raster_data.h>`

Inheritance diagram for QwtMatrixRasterData:



Public Types

- enum ResampleMode { NearestNeighbour , BilinearInterpolation , BicubicInterpolation }
  Resampling algorithm The default setting is NearestNeighbour;.

Public Member Functions

- QwtMatrixRasterData ()
  Constructor.
- virtual ~QwtMatrixRasterData ()
  Destructor.
- void setResampleMode (ResampleMode mode)
  Set the resampling algorithm.
- ResampleMode resampleMode () const
- void setInterval (Qt::Axis, const QwtInterval &)
  Assign the bounding interval for an axis.
- virtual QwtInterval interval (Qt::Axis axis) const override final
- void setValueMatrix (const QVector< double > &values, int numColumns)
  Assign a value matrix.
- const QVector< double > valueMatrix () const
- void setValue (int row, int col, double value)
  Change a single value in the matrix.
- int numColumns () const
- int numRows () const
- virtual QRectF pixelHint (const QRectF &) const override
  Calculate the pixel hint.
- virtual double value (double x, double y) const override

14.52.1 Detailed Description

A class representing a matrix of values as raster data.

QwtMatrixRasterData implements an interface for a matrix of equidistant values, that can be used by a QwtPlotRasterItem. It implements a couple of resampling algorithms, to provide values for positions, that or not on the value matrix.

Definition at line 28 of file qwt_matrix_raster_data.h.

14.52.2 Member Enumeration Documentation

14.52.2.1 ResampleMode `enum QwtMatrixRasterData::ResampleMode`

Resampling algorithm The default setting is NearestNeighbour;.

Enumerator

NearestNeighbour	Return the value from the matrix, that is nearest to the the requested position.
BilinearInterpolation	Interpolate the value from the distances and values of the 4 surrounding values in the matrix,
BicubicInterpolation	Interpolate the value from the 16 surrounding values in the matrix using hermite bicubic interpolation

Definition at line 35 of file qwt_matrix_raster_data.h.

14.52.3 Member Function Documentation

14.52.3.1 interval() `QwtInterval QwtMatrixRasterData::interval ( Qt::Axis axis ) const [final], [override], [virtual]`

Returns

Bounding interval for an axis

See also

setInterval

Implements QwtRasterData.

Definition at line 134 of file qwt_matrix_raster_data.cpp.

14.52.3.2 numColumns() `int QwtMatrixRasterData::numColumns ( ) const`

Returns

Number of columns of the value matrix

See also

valueMatrix(), numRows(), setValueMatrix()

Definition at line 195 of file qwt_matrix_raster_data.cpp.

14.52.3.3 numRows() `int QwtMatrixRasterData::numRows ( ) const`

Returns

Number of rows of the value matrix

See also

valueMatrix(), numColumns(), setValueMatrix()

Definition at line 204 of file qwt_matrix_raster_data.cpp.

14.52.3.4 pixelHint() `QRectF QwtMatrixRasterData::pixelHint ( const QRectF & area ) const [override], [virtual]`

Calculate the pixel hint.

pixelHint() returns the geometry of a pixel, that can be used to calculate the resolution and alignment of the plot item, that is representing the data.

- NearestNeighbour
  pixelHint() returns the surrounding pixel of the top left value in the matrix.
- BilinearInterpolation
  Returns an empty rectangle recommending to render in target device (f.e. screen) resolution.

Parameters

area	Requested area, ignored

Returns

Calculated hint

See also

ResampleMode, setMatrix(), setInterval()

Reimplemented from QwtRasterData.

Definition at line 229 of file qwt_matrix_raster_data.cpp.

14.52.3.5 resampleMode() QwtMatrixRasterData::ResampleMode QwtMatrixRasterData::resampleMode ( ) const

Returns

resampling algorithm

See also

setResampleMode(), value()

Definition at line 99 of file qwt_matrix_raster_data.cpp.

14.52.3.6 setInterval() void QwtMatrixRasterData::setInterval (Qt::Axis *axis*, const QwtInterval & *interval*)

Assign the bounding interval for an axis.

Setting the bounding intervals for the X/Y axis is mandatory to define the positions for the values of the value matrix. The interval in Z direction defines the possible range for the values in the matrix, what is f.e used by QwtPlotSpectrogram to map values to colors. The Z-interval might be the bounding interval of the values in the matrix, but usually it isn't. (f.e a interval of 0.0-100.0 for values in percentage)

Parameters

axis	X, Y or Z axis
interval	Interval

See also

QwtRasterData::interval(), setValueMatrix()

Definition at line 120 of file qwt_matrix_raster_data.cpp.

14.52.3.7 setResampleMode() void QwtMatrixRasterData::setResampleMode (ResampleMode *mode*)

Set the resampling algorithm.

Parameters

mode	Resampling mode

See also

resampleMode(), value()

Definition at line 90 of file qwt_matrix_raster_data.cpp.

14.52.3.8 setValue()

```
void QwtMatrixRasterData::setValue (
    int row,
    int col,
    double value )
```

Change a single value in the matrix.

Parameters

row	Row index
col	Column index
value	New value

See also

value(), setValueMatrix()

Definition at line 181 of file qwt_matrix_raster_data.cpp.

14.52.3.9 setValueMatrix()

```
void QwtMatrixRasterData::setValueMatrix (
    const QVector< double > & values,
    int numColumns )
```

Assign a value matrix.

The positions of the values are calculated by dividing the bounding rectangle of the X/Y intervals into equidistant rectangles (pixels). Each value corresponds to the center of a pixel.

Parameters

values	Vector of values
numColumns	Number of columns

See also

valueMatrix(), numColumns(), numRows(), setInterval()()

Definition at line 155 of file qwt_matrix_raster_data.cpp.

14.52.3.10 value() `double QwtMatrixRasterData::value ( double x, double y ) const [override], [virtual]`

Returns

the value at a raster position

Parameters

x	X value in plot coordinates
y	Y value in plot coordinates

See also

ResampleMode

Implements QwtRasterData.

Definition at line 256 of file qwt_matrix_raster_data.cpp.

14.52.3.11 valueMatrix() `const QVector< double > QwtMatrixRasterData::valueMatrix ( ) const`

Returns

Value matrix

See also

setValueMatrix(), numColumns(), numRows(), setInterval()

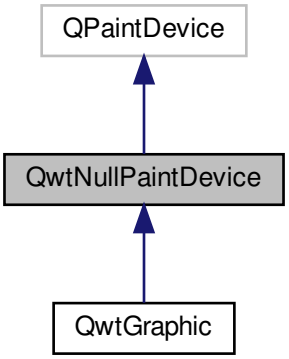
Definition at line 167 of file qwt_matrix_raster_data.cpp.

14.53 QwtNullPaintDevice Class Reference

A null paint device doing nothing.

`#include <qwt_null_paintdevice.h>`

Inheritance diagram for QwtNullPaintDevice:



Public Types

- enum Mode { NormalMode , PolygonPathMode , PathMode }

  Render mode.

Public Member Functions

- QwtNullPaintDevice ()

  Constructor.
- virtual ~QwtNullPaintDevice ()

  Destructor.
- void setMode (Mode)
- Mode mode () const
- virtual QPaintEngine * paintEngine () const override

  See QPaintDevice::paintEngine()
- virtual int metric (PaintDeviceMetric) const override
- virtual void drawRects (const QRect *, int)

  See QPaintEngine::drawRects()
- virtual void drawRects (const QRectF *, int)

  See QPaintEngine::drawRects()
- virtual void drawLines (const QLine *, int)

  See QPaintEngine::drawLines()
- virtual void drawLines (const QLineF *, int)

  See QPaintEngine::drawLines()
- virtual void drawEllipse (const QRectF &)

  See QPaintEngine::drawEllipse()
- virtual void drawEllipse (const QRect &)

  See QPaintEngine::drawEllipse()
- virtual void drawPath (const QPainterPath &)

  See QPaintEngine::drawPath()
- virtual void drawPoints (const QPointF *, int)

See QPaintEngine::drawPoints()

- virtual void drawPoints (const QPoint *, int)

  See QPaintEngine::drawPoints()
- virtual void drawPolygon (const QPointF *, int, QPaintEngine::PolygonDrawMode)

  See QPaintEngine::drawPolygon()
- virtual void drawPolygon (const QPoint *, int, QPaintEngine::PolygonDrawMode)

  See QPaintEngine::drawPolygon()
- virtual void drawPixmap (const QRectF &, const QPixmap &, const QRectF &)

  See QPaintEngine::drawPixmap()
- virtual void drawTextItem (const QPointF &, const QTextItem &)

  See QPaintEngine::drawTextItem()
- virtual void drawTiledPixmap (const QRectF &, const QPixmap &, const QPointF &)

  See QPaintEngine::drawTiledPixmap()
- virtual void drawImage (const QRectF &, const QImage &, const QRectF &, Qt::ImageConversionFlags)

  See QPaintEngine::drawImage()
- virtual void updateState (const QPaintEngineState &)

  See QPaintEngine::updateState()

Protected Member Functions

- virtual QSize sizeMetrics () const =0

14.53.1 Detailed Description

A null paint device doing nothing.

Sometimes important layout/rendering geometries are not available or changeable from the public Qt class interface. (f.e hidden in the style implementation).

QwtNullPaintDevice can be used to manipulate or filter out this information by analyzing the stream of paint primitives.

F.e. QwtNullPaintDevice is used by QwtPlotCanvas to identify styled backgrounds with rounded corners.

Definition at line 32 of file qwt_null_paintdevice.h.

14.53.2 Member Enumeration Documentation

14.53.2.1 Mode `enum QwtNullPaintDevice::Mode`

Render mode.

See also

setMode(), mode()

Enumerator

NormalMode	All vector graphic primitives are painted by the corresponding draw methods
PolygonPathMode	Vector graphic primitives (beside polygons) are mapped to a QPainterPath and are painted by drawPath. In PathMode mode only a few draw methods are called: • drawPath() • drawPixmap() • drawImage() • drawPolygon()
PathMode	Vector graphic primitives are mapped to a QPainterPath and are painted by drawPath. In PathMode mode only a few draw methods are called: • drawPath() • drawPixmap() • drawImage()

Definition at line 40 of file qwt_null_paintdevice.h.

14.53.3 Member Function Documentation

14.53.3.1 metric() `int QwtNullPaintDevice::metric ( PaintDeviceMetric deviceMetric ) const [override], [virtual]`

See QPaintDevice::metric()

Parameters

deviceMetric	Type of metric

Returns

Metric information for the given paint device metric.

See also

sizeMetrics()

Definition at line 422 of file qwt_null_paintdevice.cpp.

14.53.3.2 mode() `QwtNullPaintDevice::Mode QwtNullPaintDevice::mode ( ) const`

Returns

Render mode

See also

setMode()

Definition at line 395 of file qwt_null_paintdevice.cpp.

14.53.3.3 setMode() `void QwtNullPaintDevice::setMode ( Mode mode )`

Set the render mode

Parameters

mode	New mode

See also

mode()

Definition at line 386 of file qwt_null_paintdevice.cpp.

14.53.3.4 sizeMetrics() `virtual QSize QwtNullPaintDevice::sizeMetrics ( ) const [protected], [pure virtual]`

Returns

Size needed to implement metric()

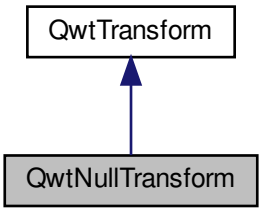
Implemented in QwtGraphic.

14.54 QwtNullTransform Class Reference

Null transformation.

`#include <qwt_transform.h>`

Inheritance diagram for QwtNullTransform:



Public Member Functions

- QwtNullTransform ()
  Constructor.
- virtual ~QwtNullTransform ()
  Destructor.
- virtual double transform (double value) const override
- virtual double invTransform (double value) const override
- virtual QwtTransform * copy () const override

14.54.1 Detailed Description

Null transformation.

QwtNullTransform returns the values unmodified.

Definition at line 80 of file qwt_transform.h.

14.54.2 Member Function Documentation

14.54.2.1 copy() `QwtTransform * QwtNullTransform::copy ( ) const [override], [virtual]`

Returns

Clone of the transformation

Implements QwtTransform.

Definition at line 68 of file qwt_transform.cpp.

14.54.2.2 invTransform() `double QwtNullTransform::invTransform ( double value ) const [override], [virtual]`

Parameters

value	Value to be transformed

Returns

value unmodified

Implements QwtTransform.

Definition at line 62 of file qwt_transform.cpp.

14.54.2.3 transform() `double QwtNullTransform::transform ( double value ) const [override], [virtual]`

Parameters

value	Value to be transformed

Returns

value unmodified

Implements QwtTransform.

Definition at line 53 of file qwt_transform.cpp.

14.55 QwtOHLCSample Class Reference

Open-High-Low-Close sample used in financial charts.

`#include <qwt_samples.h>`

Public Member Functions

- QwtOHLCSample (double time=0.0, double open=0.0, double high=0.0, double low=0.0, double close=0.0)
- QwtInterval boundingInterval () const

  Calculate the bounding interval of the OHLC values.
- bool isValid () const

  Check if a sample is valid.

Public Attributes

- double time
- double open

  Opening price.
- double high

  Highest price.
- double low

  Lowest price.
- double close

  Closing price.

14.55.1 Detailed Description

Open-High-Low-Close sample used in financial charts.

In financial charts the movement of a price in a time interval is often represented by the opening/closing prices and the lowest/highest prices in this interval.

See also

QwtTradingChartData

Definition at line 143 of file qwt_samples.h.

14.55.2 Constructor & Destructor Documentation

14.55.2.1 QwtOHLCSample()

```
QwtOHLCSample::QwtOHLCSample (
        double t = 0.0,
        double o = 0.0,
        double h = 0.0,
        double l = 0.0,
        double c = 0.0 )  [inline]
```

Constructor

Parameters

t	Time value
o	Open value
h	High value
l	Low value
c	Close value

Definition at line 182 of file qwt_samples.h.

14.55.3 Member Function Documentation

14.55.3.1 boundingInterval() `QwtInterval QwtOHLCSample::boundingInterval ( ) const [inline]`

Calculate the bounding interval of the OHLC values.

For valid samples the limits of this interval are always low/high.

Returns

Bounding interval

See also

isValid()

Definition at line 220 of file qwt_samples.h.

14.55.3.2 isValid() `bool QwtOHLCSample::isValid ( ) const [inline]`

Check if a sample is valid.

A sample is valid, when all of the following checks are true:

- low <= high
- low <= open <= high
- low <= close <= high

Returns

True, when the sample is valid

Definition at line 203 of file qwt_samples.h.

14.55.4 Member Data Documentation

14.55.4.1 time `double QwtOHLCSample::time`

Time of the sample, usually a number representing a specific interval - like a day.

Definition at line 158 of file qwt_samples.h.

14.56 QwtPainter Class Reference

A collection of QPainter workarounds.

`#include <qwt_painter.h>`

Static Public Member Functions

- static void setPolylineSplitting (bool)

  En/Disable line splitting for the raster paint engine.
- static bool polylineSplitting ()
- static void setRoundingAlignment (bool)
- static bool roundingAlignment ()
- static bool roundingAlignment (const QPainter ∗)
- static void drawText (QPainter ∗, qreal x, qreal y, const QString &)

  Wrapper for QPainter::drawText()
- static void drawText (QPainter ∗, const QPointF &, const QString &)

  Wrapper for QPainter::drawText()
- static void drawText (QPainter ∗, qreal x, qreal y, qreal w, qreal h, int flags, const QString &)

  Wrapper for QPainter::drawText()
- static void drawText (QPainter ∗, const QRectF &, int flags, const QString &)

  Wrapper for QPainter::drawText()
- static void drawSimpleRichText (QPainter ∗, const QRectF &, int flags, const QTextDocument &)
- static void drawRect (QPainter ∗, qreal x, qreal y, qreal w, qreal h)

  Wrapper for QPainter::drawRect()
- static void drawRect (QPainter ∗, const QRectF &rect)

  Wrapper for QPainter::drawRect()
- static void fillRect (QPainter ∗, const QRectF &, const QBrush &)

  Wrapper for QPainter::fillRect()
- static void drawEllipse (QPainter ∗, const QRectF &)

  Wrapper for QPainter::drawEllipse()
- static void drawPie (QPainter ∗, const QRectF &r, int a, int alen)

  Wrapper for QPainter::drawPie()
- static void drawLine (QPainter ∗, qreal x1, qreal y1, qreal x2, qreal y2)

  Wrapper for QPainter::drawLine()
- static void drawLine (QPainter ∗, const QPointF &p1, const QPointF &p2)

  Wrapper for QPainter::drawLine()
- static void drawLine (QPainter ∗, const QLineF &)

  Wrapper for QPainter::drawLine()
- static void drawPolygon (QPainter ∗, const QPolygonF &)

  Wrapper for QPainter::drawPolygon()
- static void drawPolyline (QPainter ∗, const QPolygonF &)

  Wrapper for QPainter::drawPolyline()
- static void drawPolyline (QPainter ∗, const QPointF ∗, int pointCount)

  Wrapper for QPainter::drawPolyline()
- static void drawPolygon (QPainter ∗, const QPolygon &)

  Wrapper for QPainter::drawPolygon()
- static void drawPolyline (QPainter ∗, const QPolygon &)

  Wrapper for QPainter::drawPolyline()
- static void drawPolyline (QPainter ∗, const QPoint ∗, int pointCount)

  Wrapper for QPainter::drawPolyline()

- static void drawPoint (QPainter *, const QPoint &)

  Wrapper for QPainter::drawPoint()
- static void drawPoints (QPainter *, const QPolygon &)

  Wrapper for QPainter::drawPoints()
- static void drawPoints (QPainter *, const QPoint *, int pointCount)

  Wrapper for QPainter::drawPoints()
- static void drawPoint (QPainter *, qreal x, qreal y)

  Wrapper for QPainter::drawPoint()
- static void drawPoint (QPainter *, const QPointF &)

  Wrapper for QPainter::drawPoint()
- static void drawPoints (QPainter *, const QPolygonF &)

  Wrapper for QPainter::drawPoints()
- static void drawPoints (QPainter *, const QPointF *, int pointCount)

  Wrapper for QPainter::drawPoints()
- static void drawPath (QPainter *, const QPainterPath &)

  Wrapper for QPainter::drawPath()
- static void drawImage (QPainter *, const QRectF &, const QImage &)

  Wrapper for QPainter::drawImage()
- static void drawPixmap (QPainter *, const QRectF &, const QPixmap &)

  Wrapper for QPainter::drawPixmap()
- static void drawRoundFrame (QPainter *, const QRectF &, const QPalette &, int lineWidth, int frameStyle)
- static void drawRoundedFrame (QPainter *, const QRectF &, qreal xRadius, qreal yRadius, const QPalette &, int lineWidth, int frameStyle)
- static void drawFrame (QPainter *, const QRectF &rect, const QPalette &palette, QPalette::ColorRole foregroundRole, int lineWidth, int midLineWidth, int frameStyle)
- static void drawFocusRect (QPainter *, const QWidget *)

  Draw a focus rectangle on a widget using its style.
- static void drawFocusRect (QPainter *, const QWidget *, const QRect &)

  Draw a focus rectangle on a widget using its style.
- static void drawColorBar (QPainter *, const QwtColorMap &, const QwtInterval &, const QwtScaleMap &, Qt::Orientation, const QRectF &)
- static bool isAligning (const QPainter *)
- static bool isX11GraphicsSystem ()
- static void fillPixmap (const QWidget *, QPixmap &, const QPoint &offset=QPoint())
- static void drawBackgound (QPainter *, const QRectF &, const QWidget *)
- static QPixmap backingStore (QWidget *, const QSize &)
- static qreal devicePixelRatio (const QPaintDevice *)
- static qreal effectivePenWidth (const QPen &)
- static int horizontalAdvance (const QFontMetrics &, const QString &)
- static qreal horizontalAdvance (const QFontMetricsF &, const QString &)
- static int horizontalAdvance (const QFontMetrics &, QChar)
- static qreal horizontalAdvance (const QFontMetricsF &, QChar)
- static QFont scaledFont (const QFont &, const QPaintDevice *=nullptr)

14.56.1 Detailed Description

A collection of QPainter workarounds.

Definition at line 36 of file qwt_painter.h.

14.56.2 Member Function Documentation

14.56.2.1 backingStore()

```
QPixmap QwtPainter::backingStore (
    QWidget * widget,
    const QSize & size )  [static]
```

Returns

A pixmap that can be used as backing store

Parameters

widget	Widget, for which the backingstore is intended
size	Size of the pixmap

Definition at line 1525 of file qwt_painter.cpp.

14.56.2.2 devicePixelRatio()

```
qreal QwtPainter::devicePixelRatio (
    const QPaintDevice * paintDevice )  [static]
```

Returns

Pixel ratio for a paint device

Parameters

paintDevice	Paint device

Definition at line 1491 of file qwt_painter.cpp.

14.56.2.3 drawBackgound()

```
void QwtPainter::drawBackgound (
    QPainter * painter,
    const QRectF & rect,
    const QWidget * widget )  [static]
```

Fill rect with the background of a widget

Parameters

painter	Painter
rect	Rectangle to be filled
widget	Widget

See also

QStyle::PE_Widget, QWidget::backgroundRole()

Definition at line 1351 of file qwt_painter.cpp.

14.56.2.4 drawColorBar()

```
void QwtPainter::drawColorBar (
    QPainter * painter,
    const QwtColorMap & colorMap,
    const QwtInterval & interval,
    const QwtScaleMap & scaleMap,
    Qt::Orientation orientation,
    const QRectF & rect )  [static]
```

Draw a color bar into a rectangle

Parameters

painter	Painter
colorMap	Color map
interval	Value range
scaleMap	Scale map
orientation	Orientation
rect	Target rectangle

Definition at line 1205 of file qwt_painter.cpp.

14.56.2.5 drawFrame()

```
void QwtPainter::drawFrame (
    QPainter * painter,
    const QRectF & rect,
    const QPalette & palette,
    QPalette::ColorRole foregroundRole,
    int frameWidth,
    int midLineWidth,
    int frameStyle )  [static]
```

Draw a rectangular frame

Parameters

painter	Painter
rect	Frame rectangle
palette	Palette
foregroundRole	Foreground role used for QFrame::Plain
frameWidth	Frame width
midLineWidth	Used for QFrame::Box
frameStyle	bitwise OR´ed value of QFrame::Shape and QFrame::Shadow

Definition at line 911 of file qwt_painter.cpp.

14.56.2.6 drawRoundedFrame()

```
void QwtPainter::drawRoundedFrame (
    QPainter * painter,
    const QRectF & rect,
    qreal xRadius,
    qreal yRadius,
    const QPalette & palette,
    int lineWidth,
    int frameStyle )  [static]
```

Draw a rectangular frame with rounded borders

Parameters

painter	Painter
rect	Frame rectangle
xRadius	x-radius of the ellipses defining the corners
yRadius	y-radius of the ellipses defining the corners
palette	QPalette::WindowText is used for plain borders QPalette::Dark and QPalette::Light for raised or sunken borders
lineWidth	Line width
frameStyle	bitwise OR´ed value of QFrame::Shape and QFrame::Shadow

Definition at line 1065 of file qwt_painter.cpp.

14.56.2.7 drawRoundFrame()

```
void QwtPainter::drawRoundFrame (
    QPainter * painter,
    const QRectF & rect,
    const QPalette & palette,
    int lineWidth,
    int frameStyle )  [static]
```

Draw a round frame

Parameters

painter	Painter
rect	Frame rectangle
palette	QPalette::WindowText is used for plain borders QPalette::Dark and QPalette::Light for raised or sunken borders
lineWidth	Line width
frameStyle	bitwise OR´ed value of QFrame::Shape and QFrame::Shadow

Definition at line 845 of file qwt_painter.cpp.

14.56.2.8 drawSimpleRichText()

```
void QwtPainter::drawSimpleRichText (
    QPainter * painter,
    const QRectF & rect,
    int flags,
    const QTextDocument & text ) [static]
```

Draw a text document into a rectangle

Parameters

painter	Painter
rect	Target rectangle
flags	Alignments/Text flags, see QPainter::drawText()
text	Text document

Definition at line 489 of file qwt_painter.cpp.

14.56.2.9 effectivePenWidth()

```
qreal QwtPainter::effectivePenWidth (
    const QPen & pen ) [inline], [static]
```

Returns

pen.widthF() expanded to at least 1.0

Parameters

pen	Pen

Definition at line 201 of file qwt_painter.h.

14.56.2.10 fillPixmap()

```
void QwtPainter::fillPixmap (
    const QWidget * widget,
    QPixmap & pixmap,
    const QPoint & offset = QPoint() ) [static]
```

Fill a pixmap with the content of a widget

In Qt >= 5.0 QPixmap::fill() is a nop, in Qt 4.x it is buggy for backgrounds with gradients. Thus fillPixmap() offers an alternative implementation.

Parameters

widget	Widget
pixmap	Pixmap to be filled
offset	Offset

See also

QPixmap::fill()

Definition at line 1311 of file qwt_painter.cpp.

14.56.2.11 horizontalAdvance() [1/4] int QwtPainter::horizontalAdvance (const QFontMetrics & *fontMetrics*, const QString & *text*) [static]

Distance appropriate for drawing a subsequent character after text.

Parameters

fontMetrics	Font metrics
text	Text

Returns

horizontal advance in pixels

Definition at line 1379 of file qwt_painter.cpp.

14.56.2.12 horizontalAdvance() [2/4] int QwtPainter::horizontalAdvance (const QFontMetrics & *fontMetrics*, QChar *ch*) [static]

Distance appropriate for drawing a subsequent character after ch.

Parameters

fontMetrics	Font metrics
ch	Character

Returns

horizontal advance in pixels

Definition at line 1414 of file qwt_painter.cpp.

14.56.2.13 horizontalAdvance() [3/4] qreal QwtPainter::horizontalAdvance (const QFontMetricsF & *fontMetrics*, const QString & *text*) [static]

Distance appropriate for drawing a subsequent character after text.

Parameters

fontMetrics	Font metrics
text	Text

Returns

horizontal advance in pixels

Definition at line 1397 of file qwt_painter.cpp.

14.56.2.14 horizontalAdvance() [4/4]

```
qreal QwtPainter::horizontalAdvance (
    const QFontMetricsF & fontMetrics,
    QChar ch ) [static]
```

Distance appropriate for drawing a subsequent character after ch.

Parameters

fontMetrics	Font metrics
ch	Character

Returns

horizontal advance in pixels

Definition at line 1431 of file qwt_painter.cpp.

14.56.2.15 isAligning()

```
bool QwtPainter::isAligning (
    const QPainter * painter ) [static]
```

Check if the painter is using a paint engine, that aligns coordinates to integers. Today these are all paint engines beside QPaintEngine::Pdf and QPaintEngine::SVG.

If we have an integer based paint engine it is also checked if the painter has a transformation matrix, that rotates or scales.

Parameters

painter	Painter

Returns

true, when the painter is aligning

See also

setRoundingAlignment()

Definition at line 267 of file qwt_painter.cpp.

14.56.2.16 isX11GraphicsSystem() `bool QwtPainter::isX11GraphicsSystem ( ) [static]`

Check is the application is running with the X11 graphics system that has some special capabilities that can be used for incremental painting to a widget.

Returns

True, when the graphics system is X11

Definition at line 233 of file qwt_painter.cpp.

14.56.2.17 polylineSplitting() `bool QwtPainter::polylineSplitting ( ) [inline], [static]`

Returns

True, when line splitting for the raster paint engine is enabled.

See also

setPolylineSplitting()

Definition at line 170 of file qwt_painter.h.

14.56.2.18 roundingAlignment() [1/2] `bool QwtPainter::roundingAlignment ( ) [inline], [static]`

Check whether coordinates should be rounded, before they are painted to a paint engine that rounds to integer values. For other paint engines (PDF, SVG), this flag has no effect.

Returns

True, when rounding is enabled

See also

setRoundingAlignment(), isAligning()

Definition at line 183 of file qwt_painter.h.

14.56.2.19 roundingAlignment() [2/2] `bool QwtPainter::roundingAlignment ( const QPainter * painter ) [inline], [static]`

Returns

roundingAlignment() && isAligning(painter);

Parameters

painter	Painter

Definition at line 192 of file qwt_painter.h.

14.56.2.20 scaledFont()

```
QFont QwtPainter::scaledFont (
    const QFont & font,
    const QPaintDevice * paintDevice = nullptr )  [static]
```

Adjust the DPI value of font according to the DPI value of the paint device

Parameters

font	Unscaled font
paintDevice	Paint device providing a DPI value. If paintDevice == null the DPI value of the primary screen will be used

Returns

Font being adjusted to the DPI value of the paint device

Definition at line 1450 of file qwt_painter.cpp.

14.56.2.21 setPolylineSplitting()

```
void QwtPainter::setPolylineSplitting (
    bool enable )  [static]
```

En/Disable line splitting for the raster paint engine.

In some Qt versions the raster paint engine paints polylines of many points much faster when they are split in smaller chunks: f.e all supported Qt versions >= Qt 5.0 when drawing an antialiased polyline with a pen width >=2.

Also the raster paint engine has a nasty bug in many versions (Qt 4.8 - ...) for short lines (https://codereview.qt-project.org/#/c/99456), that is worked around in this mode.

The default setting is true.

See also

polylineSplitting()

Definition at line 335 of file qwt_painter.cpp.

14.56.2.22 setRoundingAlignment() `void QwtPainter::setRoundingAlignment ( bool enable ) [static]`

Enable whether coordinates should be rounded, before they are painted to a paint engine that floors to integer values. For other paint engines (PDF, SVG) this flag has no effect. QwtPainter stores this flag only, the rounding itself is done in the painting code (f.e the plot items).

The default setting is true.

See also

roundingAlignment(), isAligning()

Definition at line 315 of file qwt_painter.cpp.

14.57 QwtPainterCommand Class Reference

`#include <qwt_painter_command.h>`

Classes

- struct **ImageData**

  Attributes how to paint a QImage.
- struct **PixmapData**

  Attributes how to paint a QPixmap.
- struct **StateData**

  Attributes of a state change.

Public Types

- enum Type {
  Invalid = -1 , Path , Pixmap , Image ,
  State }

  Type of the paint command.

Public Member Functions

- QwtPainterCommand ()

  Construct an invalid command.
- QwtPainterCommand (const QwtPainterCommand &)
- QwtPainterCommand (const QPainterPath &)

  Copy constructor.
- QwtPainterCommand (const QRectF &rect, const QPixmap &, const QRectF &subRect)
- QwtPainterCommand (const QRectF &rect, const QImage &, const QRectF &subRect, Qt::Image↩ ConversionFlags)
- QwtPainterCommand (const QPaintEngineState &)
- ~QwtPainterCommand ()

  Destructor.
- QwtPainterCommand & operator= (const QwtPainterCommand &)
- Type type () const
- QPainterPath * path ()
- const QPainterPath * path () const
- PixmapData * pixmapData ()
- const PixmapData * pixmapData () const
- ImageData * imageData ()
- const ImageData * imageData () const
- StateData * stateData ()
- const StateData * stateData () const

14.57.1 Detailed Description

QwtPainterCommand represents the attributes of a paint operation how it is used between QPainter and QPaint↩ Device

It is used by QwtGraphic to record and replay paint operations

See also

QwtGraphic::commands()

Definition at line 32 of file qwt_painter_command.h.

14.57.2 Member Enumeration Documentation

14.57.2.1 Type `enum QwtPainterCommand::Type`

Type of the paint command.

Enumerator

Invalid	Invalid command.
Path	Draw a QPainterPath.
Pixmap	Draw a QPixmap.
Image	Draw a QImage.
State	QPainter state change.

Definition at line 36 of file qwt_painter_command.h.

14.57.3 Constructor & Destructor Documentation

14.57.3.1 QwtPainterCommand() `[1/4]` `QwtPainterCommand::QwtPainterCommand ( const QwtPainterCommand & other )`

Copy constructor

Parameters

other	Command to be copied

Definition at line 128 of file qwt_painter_command.cpp.

14.57.3.2 QwtPainterCommand() [2/4]

```
QwtPainterCommand::QwtPainterCommand (
    const QRectF & rect,
    const QPixmap & pixmap,
    const QRectF & subRect )
```

Constructor for Pixmap paint operation

Parameters

rect	Target rectangle
pixmap	Pixmap
subRect	Rectangle inside the pixmap

See also

QPainter::drawPixmap()

Definition at line 34 of file qwt_painter_command.cpp.

14.57.3.3 QwtPainterCommand() [3/4]

```
QwtPainterCommand::QwtPainterCommand (
    const QRectF & rect,
    const QImage & image,
    const QRectF & subRect,
    Qt::ImageConversionFlags flags )
```

Constructor for Image paint operation

Parameters

rect	Target rectangle
image	Image
subRect	Rectangle inside the image
flags	Conversion flags

See also

QPainter::drawImage()

Definition at line 54 of file qwt_painter_command.cpp.

14.57.3.4 QwtPainterCommand() [4/4]

```
QwtPainterCommand::QwtPainterCommand (
    const QPaintEngineState & state ) [explicit]
```

Constructor for State paint operation

Parameters

state	Paint engine state

Definition at line 70 of file qwt_painter_command.cpp.

14.57.4 Member Function Documentation

14.57.4.1 imageData() [1/2] QwtPainterCommand::ImageData * QwtPainterCommand::imageData ( )

Returns

Attributes how to paint a QImage

Definition at line 228 of file qwt_painter_command.cpp.

14.57.4.2 imageData() [2/2] const QwtPainterCommand::ImageData * QwtPainterCommand::imageData ( ) const [inline]

Returns

Attributes how to paint a QImage

Definition at line 162 of file qwt_painter_command.h.

14.57.4.3 operator=() QwtPainterCommand & QwtPainterCommand::operator= (const QwtPainterCommand & *other*)

Assignment operator

Parameters

other	Command to be copied

Returns

Modified command

Definition at line 145 of file qwt_painter_command.cpp.

14.57.4.4 path() [1/2] QPainterPath * QwtPainterCommand::path ( )

Returns

Painter path to be painted

Definition at line 216 of file qwt_painter_command.cpp.

14.57.4.5 path() [2/2] const QPainterPath * QwtPainterCommand::path ( ) const [inline]

Returns

Painter path to be painted

Definition at line 148 of file qwt_painter_command.h.

14.57.4.6 pixmapData() [1/2] QwtPainterCommand::PixmapData * QwtPainterCommand::pixmapData ( )

Returns

Attributes how to paint a QPixmap

Definition at line 222 of file qwt_painter_command.cpp.

14.57.4.7 pixmapData() [2/2] const QwtPainterCommand::PixmapData * QwtPainterCommand::pixmap↩ Data ( ) const [inline]

Returns

Attributes how to paint a QPixmap

Definition at line 155 of file qwt_painter_command.h.

14.57.4.8 stateData() [1/2] QwtPainterCommand::StateData * QwtPainterCommand::stateData ( )

Returns

Attributes of a state change

Definition at line 234 of file qwt_painter_command.cpp.

14.57.4.9 stateData() [2/2] const QwtPainterCommand::StateData * QwtPainterCommand::stateData ( ) const [inline]

Returns

Attributes of a state change

Definition at line 169 of file qwt_painter_command.h.

14.57.4.10 type() QwtPainterCommand::Type QwtPainterCommand::type ( ) const [inline]

Returns

Type of the command

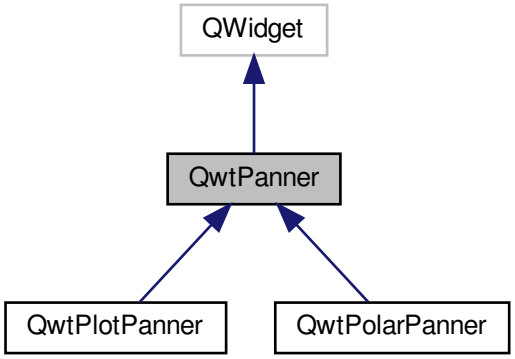
Definition at line 142 of file qwt_painter_command.h.

14.58 QwtPanner Class Reference

QwtPanner provides panning of a widget.

`#include <qwt_panner.h>`

Inheritance diagram for QwtPanner:



Signals

- void panned (int dx, int dy)
- void moved (int dx, int dy)

Public Member Functions

- QwtPanner (QWidget ∗parent)
- virtual ∼QwtPanner ()

  Destructor.
- void setEnabled (bool)

  En/disable the panner.
- bool isEnabled () const
- void setMouseButton (Qt::MouseButton, Qt::KeyboardModifiers=Qt::NoModifier)
- void getMouseButton (Qt::MouseButton &button, Qt::KeyboardModifiers &) const

  Get mouse button and modifiers used for panning.
- void setAbortKey (int key, Qt::KeyboardModifiers=Qt::NoModifier)
- void getAbortKey (int &key, Qt::KeyboardModifiers &) const

  Get the abort key and modifiers.
- void setCursor (const QCursor &)
- const QCursor cursor () const
- void setOrientations (Qt::Orientations)
- Qt::Orientations orientations () const

  Return the orientation, where panning is enabled.
- bool isOrientationEnabled (Qt::Orientation) const
- virtual bool eventFilter (QObject ∗, QEvent ∗) override

  Event filter.

Protected Member Functions

- virtual void widgetMousePressEvent (QMouseEvent ∗)
- virtual void widgetMouseReleaseEvent (QMouseEvent ∗)
- virtual void widgetMouseMoveEvent (QMouseEvent ∗)
- virtual void widgetKeyPressEvent (QKeyEvent ∗)
- virtual void widgetKeyReleaseEvent (QKeyEvent ∗)
- virtual void paintEvent (QPaintEvent ∗) override

  Paint event.
- virtual QBitmap contentsMask () const

  Calculate a mask for the contents of the panned widget.
- virtual QPixmap grab () const

14.58.1 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. 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.

Definition at line 35 of file qwt_panner.h.

14.58.2 Constructor & Destructor Documentation

14.58.2.1 QwtPanner() `QwtPanner::QwtPanner ( QWidget * parent ) [explicit]`

Creates an panner that is enabled for the left mouse button.

Parameters

parent	Parent widget to be panned

Definition at line 87 of file qwt_panner.cpp.

14.58.3 Member Function Documentation

14.58.3.1 contentsMask() `QBitmap QwtPanner::contentsMask ( ) const [protected], [virtual]`

Calculate a mask for the contents of the panned widget.

Sometimes only parts of the contents of a widget should be panned. F.e. for a widget with a styled background with rounded borders only the area inside of the border should be panned.

Returns

An empty bitmap, indicating no mask

Reimplemented in QwtPlotPanner.

Definition at line 297 of file qwt_panner.cpp.

14.58.3.2 cursor() `const QCursor QwtPanner::cursor ( ) const`

Returns

Cursor that is active while panning

See also

setCursor()

Definition at line 167 of file qwt_panner.cpp.

14.58.3.3 eventFilter() `bool QwtPanner::eventFilter ( QObject * object, QEvent * event ) [override], [virtual]`

Event filter.

When isEnabled() is true mouse events of the observed widget are filtered.

Parameters

object	Object to be filtered
event	Event

Returns

Always false, beside for paint events for the parent widget.

See also

widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseMoveEvent()

Definition at line 330 of file qwt_panner.cpp.

14.58.3.4 grab() `QPixmap QwtPanner::grab ( ) const [protected], [virtual]`

Grab the widget into a pixmap.

Returns

Grabbed pixmap

Reimplemented in QwtPlotPanner.

Definition at line 306 of file qwt_panner.cpp.

14.58.3.5 isEnabled() `bool QwtPanner::isEnabled ( ) const`

Returns

true when enabled, false otherwise

See also

setEnabled, eventFilter()

Definition at line 240 of file qwt_panner.cpp.

14.58.3.6 isOrientationEnabled() `bool QwtPanner::isOrientationEnabled ( Qt::Orientation o ) const`

Returns

True if an orientation is enabled

See also

orientations(), setOrientations()

Definition at line 231 of file qwt_panner.cpp.

14.58.3.7 moved `void QwtPanner::moved ( int dx, int dy ) [signal]`

Signal emitted, while the widget moved, but panning is not finished.

Parameters

dx	Offset in horizontal direction
dy	Offset in vertical direction

14.58.3.8 paintEvent() `void QwtPanner::paintEvent ( QPaintEvent * event ) [override], [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

event	Paint event

Definition at line 253 of file qwt_panner.cpp.

14.58.3.9 panned `void QwtPanner::panned ( int dx, int dy ) [signal]`

Signal emitted, when panning is done

Parameters

dx	Offset in horizontal direction
dy	Offset in vertical direction

14.58.3.10 setAbortKey() `void QwtPanner::setAbortKey ( int key, Qt::KeyboardModifiers modifiers = Qt::NoModifier )`

Change the abort key The defaults are Qt::Key_Escape and Qt::NoModifiers

Parameters

key	Key (See Qt::Keycode)
modifiers	Keyboard modifiers

Definition at line 132 of file qwt_panner.cpp.

14.58.3.11 setCursor() void QwtPanner::setCursor (const QCursor & *cursor*)

Change the cursor, that is active while panning The default is the cursor of the parent widget.

Parameters

cursor	New cursor

See also

setCursor()

Definition at line 156 of file qwt_panner.cpp.

14.58.3.12 setEnabled() 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

on	true or false

See also

isEnabled(), eventFilter()

Definition at line 188 of file qwt_panner.cpp.

14.58.3.13 setMouseButton() void QwtPanner::setMouseButton (Qt::MouseButton *button*, Qt::KeyboardModifiers *modifiers* = *Qt::NoModifier*)

Change the mouse button and modifiers used for panning The defaults are Qt::LeftButton and Qt::NoModifier

Definition at line 110 of file qwt_panner.cpp.

14.58.3.14 setOrientations() 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

Definition at line 216 of file qwt_panner.cpp.

14.58.3.15 widgetKeyPressEvent() `void QwtPanner::widgetKeyPressEvent ( QKeyEvent * keyEvent ) [protected], [virtual]`

Handle a key press event for the observed widget.

Parameters

keyEvent	Key event

See also

eventFilter(), widgetKeyReleaseEvent()

Definition at line 482 of file qwt_panner.cpp.

14.58.3.16 widgetKeyReleaseEvent() `void QwtPanner::widgetKeyReleaseEvent ( QKeyEvent * keyEvent ) [protected], [virtual]`

Handle a key release event for the observed widget.

Parameters

keyEvent	Key event

See also

eventFilter(), widgetKeyReleaseEvent()

Definition at line 502 of file qwt_panner.cpp.

14.58.3.17 widgetMouseMoveEvent() `void QwtPanner::widgetMouseMoveEvent ( QMouseEvent * mouseEvent ) [protected], [virtual]`

Handle a mouse move event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent()

Definition at line 421 of file qwt_panner.cpp.

14.58.3.18 widgetMousePressEvent() `void QwtPanner::widgetMousePressEvent ( QMouseEvent * mouseEvent ) [protected], [virtual]`

Handle a mouse press event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMouseReleaseEvent(), widgetMouseMoveEvent(),

Reimplemented in QwtPolarPanner.

Definition at line 381 of file qwt_panner.cpp.

14.58.3.19 widgetMouseReleaseEvent() `void QwtPanner::widgetMouseReleaseEvent ( QMouseEvent * mouseEvent ) [protected], [virtual]`

Handle a mouse release event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseMoveEvent(),

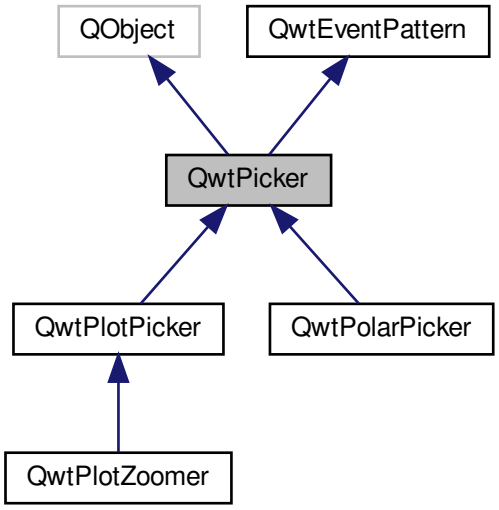
Definition at line 449 of file qwt_panner.cpp.

14.59 QwtPicker Class Reference

QwtPicker provides selections on a widget.

```
#include <qwt_picker.h>
```

Inheritance diagram for QwtPicker:



Public Types

- enum RubberBand {
  NoRubberBand = 0 , HLineRubberBand , VLineRubberBand , CrossRubberBand ,
  RectRubberBand , EllipseRubberBand , PolygonRubberBand , UserRubberBand = 100 }
- enum DisplayMode { AlwaysOff , AlwaysOn , ActiveOnly }

  Display mode.
- enum ResizeMode { Stretch , KeepSize }

Public Slots

- void setEnabled (bool)

  En/disable the picker.

Signals

- void activated (bool on)
- void selected (const QPolygon &polygon)
- void appended (const QPoint &pos)
- void moved (const QPoint &pos)
- void removed (const QPoint &pos)
- void changed (const QPolygon &selection)

Public Member Functions

- QwtPicker (QWidget *parent)
- QwtPicker (RubberBand rubberBand, DisplayMode trackerMode, QWidget *)
- virtual ~QwtPicker ()

  Destructor.
- void setStateMachine (QwtPickerMachine *)
- const QwtPickerMachine * stateMachine () const
- QwtPickerMachine * stateMachine ()
- void setRubberBand (RubberBand)
- RubberBand rubberBand () const
- void setTrackerMode (DisplayMode)

  Set the display mode of the tracker.
- DisplayMode trackerMode () const
- void setResizeMode (ResizeMode)

  Set the resize mode.
- ResizeMode resizeMode () const
- void setRubberBandPen (const QPen &)
- QPen rubberBandPen () const
- void setTrackerPen (const QPen &)
- QPen trackerPen () const
- void setTrackerFont (const QFont &)
- QFont trackerFont () const
- bool isEnabled () const
- bool isActive () const
- virtual bool eventFilter (QObject *, QEvent *) override

  Event filter.
- QWidget * parentWidget ()

  Return the parent widget, where the selection happens.
- const QWidget * parentWidget () const

  Return the parent widget, where the selection happens.
- virtual QPainterPath pickArea () const
- virtual void drawRubberBand (QPainter *) const
- virtual void drawTracker (QPainter *) const
- virtual QRegion trackerMask () const
- virtual QRegion rubberBandMask () const
- virtual QwtText trackerText (const QPoint &pos) const

  Return the label for a position.
- QPoint trackerPosition () const
- virtual QRect trackerRect (const QFont &) const
- QPolygon selection () const

Protected Member Functions

- virtual QPolygon adjustedPoints (const QPolygon &) const

  Map the pickedPoints() into a selection()
- virtual void transition (const QEvent *)
- virtual void begin ()
- virtual void append (const QPoint &)
- virtual void move (const QPoint &)
- virtual void remove ()
- virtual bool end (bool ok=true)

Close a selection setting the state to inactive.

- virtual bool accept (QPolygon &) const

  Validate and fix up the selection.
- virtual void reset ()
- virtual void widgetMousePressEvent (QMouseEvent *)
- virtual void widgetMouseReleaseEvent (QMouseEvent *)
- virtual void widgetMouseDoubleClickEvent (QMouseEvent *)
- virtual void widgetMouseMoveEvent (QMouseEvent *)
- virtual void widgetWheelEvent (QWheelEvent *)
- virtual void widgetKeyPressEvent (QKeyEvent *)
- virtual void widgetKeyReleaseEvent (QKeyEvent *)
- virtual void widgetEnterEvent (QEvent *)
- virtual void widgetLeaveEvent (QEvent *)
- virtual void stretchSelection (const QSize &oldSize, const QSize &newSize)
- virtual void updateDisplay ()

  Update the state of rubber band and tracker label.
- const QwtWidgetOverlay * rubberBandOverlay () const
- const QwtWidgetOverlay * trackerOverlay () const
- const QPolygon & pickedPoints () const

14.59.1 Detailed Description

QwtPicker provides selections on a widget.

QwtPicker filters all enter, leave, mouse and keyboard events of a widget and translates them into an array of selected points.

The way how the points are collected depends on type of state machine that is connected to the picker. Qwt offers a couple of predefined state machines for selecting:

- Nothing
  QwtPickerTrackerMachine
- Single points
  QwtPickerClickPointMachine, QwtPickerDragPointMachine
- Rectangles
  QwtPickerClickRectMachine, QwtPickerDragRectMachine
- Polygons
  QwtPickerPolygonMachine

While these state machines cover the most common ways to collect points it is also possible to implement individual machines as well.

QwtPicker translates the picked points into a selection using the adjustedPoints() method. adjustedPoints() is intended to be reimplemented to fix up the selection according to application specific requirements. (F.e. when an application accepts rectangles of a fixed aspect ratio only.)

Optionally QwtPicker support the process of collecting points by a rubber band and tracker displaying a text for the current mouse position.

Example

```
#include <qwt_picker.h>
#include <qwt_picker_machine.h>
   QwtPicker *picker = new QwtPicker(widget);
   picker->setStateMachine(new QwtPickerDragRectMachine);
   picker->setTrackerMode(QwtPicker::ActiveOnly);
   picker->setRubberBand(QwtPicker::RectRubberBand);
```

The state machine triggers the following commands:

- begin()
  Activate/Initialize the selection.
- append()
  Add a new point
- move()
  Change the position of the last point.
- remove()
  Remove the last point.
- end()
  Terminate the selection and call accept to validate the picked points.

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 while the picker is active, or if trackerMode() is AlwayOn.

Definition at line 103 of file qwt_picker.h.

14.59.2 Member Enumeration Documentation

14.59.2.1 DisplayMode `enum QwtPicker::DisplayMode`

Display mode.

See also

setTrackerMode(), trackerMode(), isActive()

Enumerator

AlwaysOff	Display never.
AlwaysOn	Display always.
ActiveOnly	Display only when the selection is active.

Definition at line 161 of file qwt_picker.h.

14.59.2.2 ResizeMode enum QwtPicker::ResizeMode

Controls what to do with the selected points of an active selection when the observed widget is resized.

The default value is QwtPicker::Stretch.

See also

setResizeMode()

Enumerator

Stretch	All points are scaled according to the new size,.
KeepSize	All points remain unchanged.

Definition at line 181 of file qwt_picker.h.

14.59.2.3 RubberBand enum QwtPicker::RubberBand

Rubber band style

The default value is QwtPicker::NoRubberBand.

See also

setRubberBand(), rubberBand()

Enumerator

NoRubberBand	No rubberband.
HLineRubberBand	A horizontal line (only for QwtPickerMachine::PointSelection)
VLineRubberBand	A vertical line (only for QwtPickerMachine::PointSelection)
CrossRubberBand	A crosshair (only for QwtPickerMachine::PointSelection)
RectRubberBand	A rectangle (only for QwtPickerMachine::RectSelection)
EllipseRubberBand	An ellipse (only for QwtPickerMachine::RectSelection)
PolygonRubberBand	A polygon (only for QwtPickerMachine::PolygonSelection)
UserRubberBand	Values >= UserRubberBand can be used to define additional rubber bands.

Definition at line 127 of file qwt_picker.h.

14.59.3 Constructor & Destructor Documentation

14.59.3.1 QwtPicker() [1/2] QwtPicker::QwtPicker (QWidget * *parent*) [explicit]

Constructor

Creates an picker that is enabled, but without a state machine. rubber band and tracker are disabled.

Parameters

parent	Parent widget, that will be observed

Definition at line 168 of file qwt_picker.cpp.

14.59.3.2 QwtPicker() [2/2] QwtPicker::QwtPicker (RubberBand *rubberBand*, DisplayMode *trackerMode*, QWidget * *parent*) [explicit]

Constructor

Parameters

rubberBand	Rubber band style
trackerMode	Tracker mode
parent	Parent widget, that will be observed

Definition at line 181 of file qwt_picker.cpp.

14.59.4 Member Function Documentation

14.59.4.1 accept() bool QwtPicker::accept (QPolygon & *selection*) const [protected], [virtual]

Validate and fix up the selection.

Accepts all selections unmodified

Parameters

selection	Selection to validate and fix up

Returns

true, when accepted, false otherwise

Reimplemented in QwtPlotZoomer.

Definition at line 1383 of file qwt_picker.cpp.

14.59.4.2 activated

```
void QwtPicker::activated (
        bool on )  [signal]
```

A signal indicating, when the picker has been activated. Together with setEnabled() it can be used to implement selections with more than one picker.

Parameters

on	True, when the picker has been activated

14.59.4.3 adjustedPoints()

```
QPolygon QwtPicker::adjustedPoints (
        const QPolygon & points ) const  [protected], [virtual]
```

Map the pickedPoints() into a selection()

adjustedPoints() maps the points, that have been collected on the parentWidget() into a selection(). The default implementation simply returns the points unmodified.

The reason, why a selection() differs from the picked points depends on the application requirements. F.e. :

- A rectangular selection might need to have a specific aspect ratio only.
- A selection could accept non intersecting polygons only.
- ...

The example below is for a rectangular selection, where the first point is the center of the selected rectangle.

Example

```
QPolygon MyPicker::adjustedPoints( const QPolygon &points ) const
{
    QPolygon adjusted;
    if ( points.size() == 2 )
    {
        const int width = qAbs( points[1].x() - points[0].x() );
        const int height = qAbs( points[1].y() - points[0].y() );
        QRect rect( 0, 0, 2 * width, 2 * height );
        rect.moveCenter( points[0] );
        adjusted += rect.topLeft();
        adjusted += rect.bottomRight();
    }
    return adjusted;
}
```

Parameters

points	Selected points

Returns

Selected points unmodified

Definition at line 783 of file qwt_picker.cpp.

14.59.4.4 append() `void QwtPicker::append ( const QPoint & pos ) [protected], [virtual]`

Append a point to the selection and update rubber band and tracker. The appended() signal is emitted.

Parameters

pos	Additional point

See also

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

Reimplemented in QwtPolarPicker, and QwtPlotPicker.

Definition at line 1320 of file qwt_picker.cpp.

14.59.4.5 appended `void QwtPicker::appended ( const QPoint & pos ) [signal]`

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

Parameters

pos	Position of the appended point.

See also

append(). moved()

14.59.4.6 begin() `void QwtPicker::begin ( ) [protected], [virtual]`

Open a selection setting the state to active

See also

isActive(), end(), append(), move()

Reimplemented in QwtPlotZoomer.

Definition at line 1239 of file qwt_picker.cpp.

14.59.4.7 changed `void QwtPicker::changed ( const QPolygon & selection ) [signal]`

A signal emitted when the active selection has been changed. This might happen when the observed widget is resized.

Parameters

selection	Changed selection

See also

stretchSelection()

14.59.4.8 drawRubberBand() `void QwtPicker::drawRubberBand ( QPainter * painter ) const [virtual]`

Draw a rubber band, depending on rubberBand()

Parameters

painter	Painter, initialized with a clip region

See also

rubberBand(), RubberBand

Definition at line 637 of file qwt_picker.cpp.

14.59.4.9 drawTracker() `void QwtPicker::drawTracker ( QPainter * painter ) const [virtual]`

Draw the tracker

Parameters

painter	Painter

See also

trackerRect(), trackerText()

Definition at line 732 of file qwt_picker.cpp.

14.59.4.10 end() bool QwtPicker::end (bool ok = true) [protected], [virtual]

Close a selection setting the state to inactive.

The selection is validated and maybe fixed by accept().

Parameters

ok	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 QwtPolarPicker, QwtPlotZoomer, and QwtPlotPicker.

Definition at line 1272 of file qwt_picker.cpp.

14.59.4.11 eventFilter() bool QwtPicker::eventFilter (QObject * object, QEvent * event) [override], [virtual]

Event filter.

When isEnabled() is 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 rubber band and tracker up to date.

Parameters

object	Object to be filtered
event	Event

Returns

Always false.

See also

widgetEnterEvent(), widgetLeaveEvent(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEv
widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent(),
QObject::installEventFilter(), QObject::event()

Definition at line 893 of file qwt_picker.cpp.

14.59.4.12 isActive() `bool QwtPicker::isActive ( ) const`

A picker is active between begin() and end().

Returns

true if the selection is active.

Definition at line 1393 of file qwt_picker.cpp.

14.59.4.13 isEnabled() `bool QwtPicker::isEnabled ( ) const`

Returns

true when enabled, false otherwise

See also

setEnabled(), eventFilter()

Definition at line 399 of file qwt_picker.cpp.

14.59.4.14 move() `void QwtPicker::move ( const QPoint & pos ) [protected], [virtual]`

Move the last point of the selection The moved() signal is emitted.

Parameters

pos	New position

See also

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

Reimplemented in QwtPolarPicker, and QwtPlotPicker.

Definition at line 1338 of file qwt_picker.cpp.

14.59.4.15 moved `void QwtPicker::moved ( const QPoint & pos ) [signal]`

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

Parameters

pos	Position of the moved last point of the selection.

See also

move(), appended()

14.59.4.16 pickArea() `QPainterPath QwtPicker::pickArea ( ) const [virtual]`

Find the area of the observed widget, where selection might happen.

Returns

parentWidget()->contentsRect()

Reimplemented in QwtPolarPicker.

Definition at line 1474 of file qwt_picker.cpp.

14.59.4.17 pickedPoints() `const QPolygon & QwtPicker::pickedPoints ( ) const [protected]`

Return the points, that have been collected so far. The selection() is calculated from the pickedPoints() in adjustedPoints().

Returns

Picked points

Definition at line 1403 of file qwt_picker.cpp.

14.59.4.18 remove() `void QwtPicker::remove ( ) [protected], [virtual]`

Remove the last point of the selection The removed() signal is emitted.

See also

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

Definition at line 1359 of file qwt_picker.cpp.

14.59.4.19 removed `void QwtPicker::removed ( const QPoint & pos ) [signal]`

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

Parameters

pos	Position of the point, that has been removed

See also

remove(), appended()

14.59.4.20 reset() `void QwtPicker::reset ( ) [protected], [virtual]`

Reset the state machine and terminate (end(false)) the selection

Definition at line 1303 of file qwt_picker.cpp.

14.59.4.21 resizeMode() `QwtPicker::ResizeMode QwtPicker::resizeMode ( ) const`

Returns

Resize mode

See also

setResizeMode(), ResizeMode

Definition at line 361 of file qwt_picker.cpp.

14.59.4.22 rubberBand() `QwtPicker::RubberBand QwtPicker::rubberBand ( ) const`

Returns

Rubber band style

See also

setRubberBand(), RubberBand, rubberBandPen()

Definition at line 298 of file qwt_picker.cpp.

14.59.4.23 rubberBandMask() `QRegion QwtPicker::rubberBandMask ( ) const [virtual]`

Calculate the mask for the rubber band overlay

Returns

Region for the mask

See also

QWidget::setMask()

Definition at line 525 of file qwt_picker.cpp.

14.59.4.24 rubberBandOverlay() `const QwtWidgetOverlay * QwtPicker::rubberBandOverlay ( ) const [protected]`

Returns

Overlay displaying the rubber band

Definition at line 1581 of file qwt_picker.cpp.

14.59.4.25 rubberBandPen() `QPen QwtPicker::rubberBandPen ( ) const`

Returns

Rubber band pen

See also

setRubberBandPen(), rubberBand()

Definition at line 472 of file qwt_picker.cpp.

14.59.4.26 selected `void QwtPicker::selected ( const QPolygon & polygon ) [signal]`

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

Parameters

polygon	Selected points

14.59.4.27 selection() `QPolygon QwtPicker::selection ( ) const`

Returns

Selected points

See also

pickedPoints(), adjustedPoints()

Definition at line 792 of file qwt_picker.cpp.

14.59.4.28 setEnabled `void QwtPicker::setEnabled ( bool enabled ) [slot]`

En/disable the picker.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

enabled	true or false

See also

isEnabled(), eventFilter()

Definition at line 375 of file qwt_picker.cpp.

14.59.4.29 setResizeMode() `void QwtPicker::setResizeMode ( ResizeMode mode )`

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

mode	Resize mode

See also

resizeMode(), ResizeMode

Definition at line 351 of file qwt_picker.cpp.

14.59.4.30 setRubberBand() `void QwtPicker::setRubberBand ( RubberBand rubberBand )`

Set the rubber band style

Parameters

rubberBand	Rubber band style The default value is NoRubberBand.

See also

rubberBand(), RubberBand, setRubberBandPen()

Definition at line 289 of file qwt_picker.cpp.

14.59.4.31 setRubberBandPen() `void QwtPicker::setRubberBandPen ( const QPen & pen )`

Set the pen for the rubberband

Parameters

pen	Rubber band pen

See also

rubberBandPen(), setRubberBand()

Definition at line 459 of file qwt_picker.cpp.

14.59.4.32 setStateMachine() `void QwtPicker::setStateMachine ( QwtPickerMachine * stateMachine )`

Set a state machine and delete the previous one

Parameters

stateMachine	State machine

See also

stateMachine()

Definition at line 229 of file qwt_picker.cpp.

14.59.4.33 setTrackerFont() `void QwtPicker::setTrackerFont ( const QFont & font )`

Set the font for the tracker

Parameters

font	Tracker font

See also

trackerFont(), setTrackerMode(), setTrackerPen()

Definition at line 410 of file qwt_picker.cpp.

14.59.4.34 setTrackerMode() `void QwtPicker::setTrackerMode ( DisplayMode mode )`

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

mode	Tracker display mode

Warning

In case of AlwaysOn, mouseTracking will be enabled for the observed widget.

See also

trackerMode(), DisplayMode

Definition at line 319 of file qwt_picker.cpp.

14.59.4.35 setTrackerPen() void QwtPicker::setTrackerPen (const QPen & *pen*)

Set the pen for the tracker

Parameters

pen	Tracker pen

See also

trackerPen(), setTrackerMode(), setTrackerFont()

Definition at line 435 of file qwt_picker.cpp.

14.59.4.36 stateMachine() [1/2] QwtPickerMachine * QwtPicker::stateMachine ( )

Returns

Assigned state machine

See also

setStateMachine()

Definition at line 247 of file qwt_picker.cpp.

14.59.4.37 stateMachine() [2/2] const QwtPickerMachine * QwtPicker::stateMachine ( ) const

Returns

Assigned state machine

See also

setStateMachine()

Definition at line 256 of file qwt_picker.cpp.

14.59.4.38 stretchSelection()

```
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

oldSize	Previous size
newSize	Current size

See also

ResizeMode, setResizeMode(), resizeMode()

Definition at line 1417 of file qwt_picker.cpp.

14.59.4.39 trackerFont() `QFont QwtPicker::trackerFont ( ) const`

Returns

Tracker font

See also

setTrackerFont(), trackerMode(), trackerPen()

Definition at line 424 of file qwt_picker.cpp.

14.59.4.40 trackerMask() `QRegion QwtPicker::trackerMask ( ) const [virtual]`

Calculate the mask for the tracker overlay

Returns

Region with one rectangle: trackerRect(trackerFont());

See also

QWidget::setMask(), trackerRect()

Definition at line 514 of file qwt_picker.cpp.

14.59.4.41 trackerMode() `QwtPicker::DisplayMode QwtPicker::trackerMode ( ) const`

Returns

Tracker display mode

See also

setTrackerMode(), DisplayMode

Definition at line 332 of file qwt_picker.cpp.

14.59.4.42 trackerOverlay() `const QwtWidgetOverlay * QwtPicker::trackerOverlay ( ) const [protected]`

Returns

Overlay displaying the tracker text

Definition at line 1587 of file qwt_picker.cpp.

14.59.4.43 trackerPen() `QPen QwtPicker::trackerPen ( ) const`

Returns

Tracker pen

See also

setTrackerPen(), trackerMode(), trackerFont()

Definition at line 448 of file qwt_picker.cpp.

14.59.4.44 trackerPosition() `QPoint QwtPicker::trackerPosition ( ) const`

Returns

Current position of the tracker

Definition at line 798 of file qwt_picker.cpp.

14.59.4.45 trackerRect() `QRect QwtPicker::trackerRect ( const QFont & font ) const [virtual]`

Calculate the bounding rectangle for the tracker text from the current position of the tracker

Parameters

font	Font of the tracker text

Returns

Bounding rectangle of the tracker text

See also

trackerPosition()

Definition at line 812 of file qwt_picker.cpp.

14.59.4.46 trackerText() `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

pos	Position

Returns

Converted position as string

Reimplemented in QwtPolarPicker, and QwtPlotPicker.

Definition at line 490 of file qwt_picker.cpp.

14.59.4.47 transition() `void QwtPicker::transition ( const QEvent * event ) [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

event	Event

Definition at line 1176 of file qwt_picker.cpp.

14.59.4.48 widgetEnterEvent() `void QwtPicker::widgetEnterEvent ( QEvent * event ) [protected], [virtual]`

Handle a enter event for the observed widget.

Parameters

event	Qt event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Definition at line 1018 of file qwt_picker.cpp.

14.59.4.49 widgetKeyPressEvent() `void QwtPicker::widgetKeyPressEvent ( QKeyEvent * keyEvent ) [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.

Parameters

keyEvent	Key event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyReleaseEvent(), stateMachine(), QwtEventPattern::KeyPatternCode

Reimplemented in QwtPlotZoomer.

Definition at line 1112 of file qwt_picker.cpp.

14.59.4.50 widgetKeyReleaseEvent() `void QwtPicker::widgetKeyReleaseEvent ( QKeyEvent * keyEvent ) [protected], [virtual]`

Handle a key release event for the observed widget.

Passes the event to the state machine.

Parameters

keyEvent	Key event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), stateMachine()

Definition at line 1164 of file qwt_picker.cpp.

14.59.4.51 widgetLeaveEvent()

```
void QwtPicker::widgetLeaveEvent (
    QEvent * event )  [protected], [virtual]
```

Handle a leave event for the observed widget.

Parameters

event	Qt event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Definition at line 1032 of file qwt_picker.cpp.

14.59.4.52 widgetMouseDoubleClickEvent()

```
void QwtPicker::widgetMouseDoubleClickEvent (
    QMouseEvent * mouseEvent )  [protected], [virtual]
```

Handle mouse double click event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Definition at line 1064 of file qwt_picker.cpp.

14.59.4.53 widgetMouseMoveEvent()

```
void QwtPicker::widgetMouseMoveEvent (
    QMouseEvent * mouseEvent )  [protected], [virtual]
```

Handle a mouse move event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Definition at line 996 of file qwt_picker.cpp.

14.59.4.54 widgetMousePressEvent()

```
void QwtPicker::widgetMousePressEvent (
    QMouseEvent * mouseEvent ) [protected], [virtual]
```

Handle a mouse press event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Definition at line 982 of file qwt_picker.cpp.

14.59.4.55 widgetMouseReleaseEvent()

```
void QwtPicker::widgetMouseReleaseEvent (
    QMouseEvent * mouseEvent ) [protected], [virtual]
```

Handle a mouse release event for the observed widget.

Parameters

mouseEvent	Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Reimplemented in QwtPlotZoomer.

Definition at line 1050 of file qwt_picker.cpp.

14.59.4.56 widgetWheelEvent() `void QwtPicker::widgetWheelEvent ( QWheelEvent * wheelEvent ) [protected], [virtual]`

Handle a wheel event for the observed widget.

Move the last point of the selection in case of isActive() == true

Parameters

wheelEvent	Wheel event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

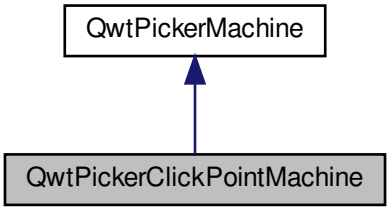
Definition at line 1081 of file qwt_picker.cpp.

14.60 QwtPickerClickPointMachine Class Reference

A state machine for point selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerClickPointMachine:



Public Member Functions

- QwtPickerClickPointMachine ()

  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override

  Transition.

Additional Inherited Members

14.60.1 Detailed Description

A state machine for point selections.

Pressing QwtEventPattern::MouseSelect1 or QwtEventPattern::KeySelect1 selects a point.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

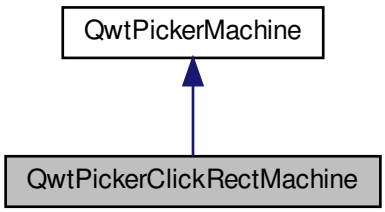
Definition at line 102 of file qwt_picker_machine.h.

14.61 QwtPickerClickRectMachine Class Reference

A state machine for rectangle selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerClickRectMachine:



Public Member Functions

- QwtPickerClickRectMachine ()
  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override
  Transition.

Additional Inherited Members

14.61.1 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

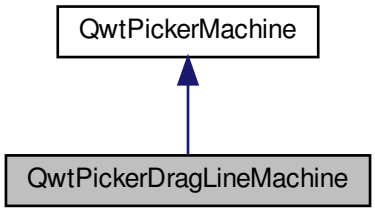
Definition at line 140 of file qwt_picker_machine.h.

14.62 QwtPickerDragLineMachine Class Reference

A state machine for line selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerDragLineMachine:



Public Member Functions

- QwtPickerDragLineMachine ()
  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override
  Transition.

Additional Inherited Members

14.62.1 Detailed Description

A state machine for line 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.

A common use case of QwtPickerDragLineMachine are pickers for distance measurements.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

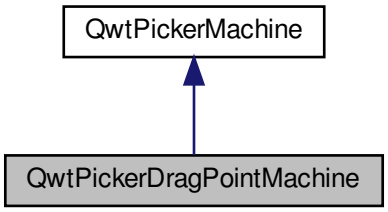
Definition at line 185 of file qwt_picker_machine.h.

14.63 QwtPickerDragPointMachine Class Reference

A state machine for point selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerDragPointMachine:



Public Member Functions

- QwtPickerDragPointMachine ()
  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override
  Transition.

Additional Inherited Members

14.63.1 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.

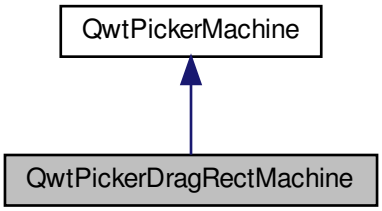
Definition at line 118 of file qwt_picker_machine.h.

14.64 QwtPickerDragRectMachine Class Reference

A state machine for rectangle selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerDragRectMachine:



Public Member Functions

- QwtPickerDragRectMachine ()

  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override

  Transition.

Additional Inherited Members

14.64.1 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

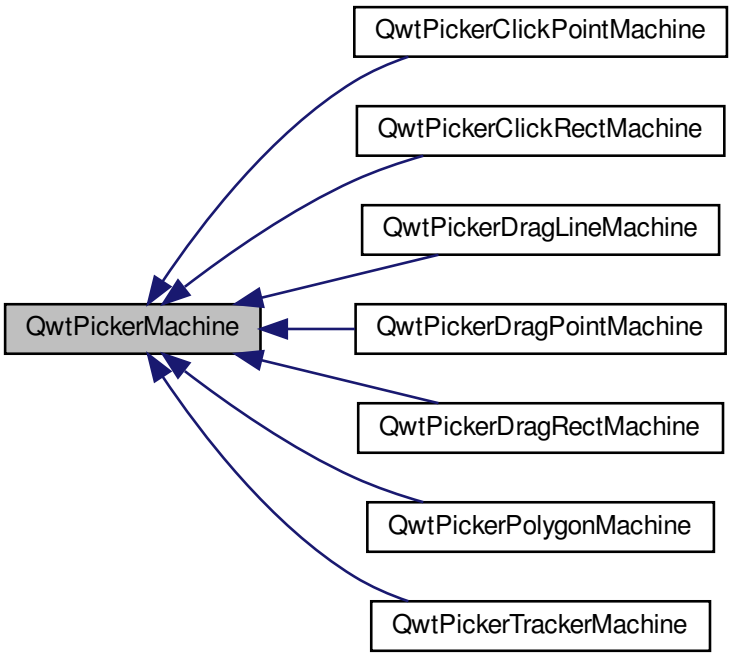
Definition at line 161 of file qwt_picker_machine.h.

14.65 QwtPickerMachine Class Reference

A state machine for QwtPicker selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerMachine:



Public Types

- enum SelectionType { NoSelection = -1 , PointSelection , RectSelection , PolygonSelection }
- enum Command {
  Begin , **Append** , **Move** , **Remove** ,
  End }

  Commands - the output of a state machine.

Public Member Functions

- QwtPickerMachine (SelectionType)

  Constructor.
- virtual ~QwtPickerMachine ()

  Destructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *)=0

  Transition.
- void reset ()

  Set the current state to 0.
- int state () const

  Return the current state.
- void setState (int)

  Change the current state.
- SelectionType selectionType () const

  Return the selection type.

14.65.1 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

Definition at line 28 of file qwt_picker_machine.h.

14.65.2 Member Enumeration Documentation

14.65.2.1 SelectionType `enum QwtPickerMachine::SelectionType`

Type of a selection.

See also

selectionType()

Enumerator

NoSelection	The state machine not usable for any type of selection.
PointSelection	The state machine is for selecting a single point.
RectSelection	The state machine is for selecting a rectangle (2 points).
PolygonSelection	The state machine is for selecting a polygon (many points).

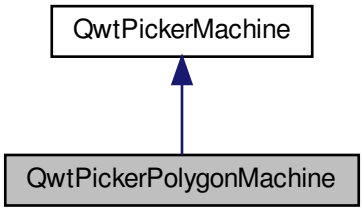
Definition at line 35 of file qwt_picker_machine.h.

14.66 QwtPickerPolygonMachine Class Reference

A state machine for polygon selections.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerPolygonMachine:



Public Member Functions

- QwtPickerPolygonMachine ()

  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override

  Transition.

Additional Inherited Members

14.66.1 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

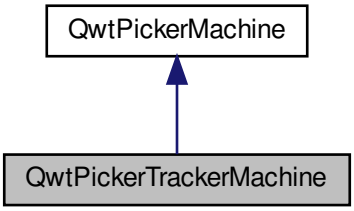
Definition at line 205 of file qwt_picker_machine.h.

14.67 QwtPickerTrackerMachine Class Reference

A state machine for indicating mouse movements.

```
#include <qwt_picker_machine.h>
```

Inheritance diagram for QwtPickerTrackerMachine:



Public Member Functions

- QwtPickerTrackerMachine ()
  Constructor.
- virtual QList< Command > transition (const QwtEventPattern &, const QEvent *) override
  Transition.

Additional Inherited Members

14.67.1 Detailed Description

A state machine for indicating mouse movements.

QwtPickerTrackerMachine supports displaying information corresponding to mouse movements, but is not intended for selecting anything. Begin/End are related to Enter/Leave events.

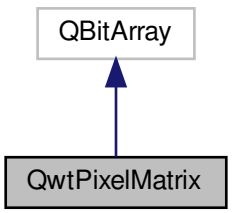
Definition at line 85 of file qwt_picker_machine.h.

14.68 QwtPixelMatrix Class Reference

A bit field corresponding to the pixels of a rectangle.

`#include <qwt_pixel_matrix.h>`

Inheritance diagram for QwtPixelMatrix:



Public Member Functions

- QwtPixelMatrix (const QRect &rect)

  Constructor.
- ~QwtPixelMatrix ()

  Destructor.
- void setRect (const QRect &rect)
- QRect rect () const
- bool testPixel (int x, int y) const

  Test if a pixel has been set.
- bool testAndSetPixel (int x, int y, bool on)

  Set a pixel and test if a pixel has been set before.

Public Attributes

- int **const**

14.68.1 Detailed Description

A bit field corresponding to the pixels of a rectangle.

QwtPixelMatrix is intended to filter out duplicates in an unsorted array of points.

Definition at line 24 of file qwt_pixel_matrix.h.

14.68.2 Constructor & Destructor Documentation

14.68.2.1 QwtPixelMatrix() `QwtPixelMatrix::QwtPixelMatrix ( const QRect & rect ) [explicit]`

Constructor.

Parameters

rect	Bounding rectangle for the matrix

Definition at line 17 of file qwt_pixel_matrix.cpp.

14.68.3 Member Function Documentation

14.68.3.1 rect() `QRect QwtPixelMatrix::rect ( ) const`

Returns

Bounding rectangle

Definition at line 48 of file qwt_pixel_matrix.cpp.

14.68.3.2 setRect() `void QwtPixelMatrix::setRect ( const QRect & rect )`

Set the bounding rectangle of the matrix

Parameters

rect	Bounding rectangle

Note

All bits are cleared

Definition at line 35 of file qwt_pixel_matrix.cpp.

14.68.3.3 testAndSetPixel() `bool QwtPixelMatrix::testAndSetPixel ( int x, int y, bool on ) [inline]`

Set a pixel and test if a pixel has been set before.

Parameters

x	X-coordinate
y	Y-coordinate
on	Set/Clear the pixel

Returns

true, when pos is outside of rect(), or when the pixel was set before.

Definition at line 67 of file qwt_pixel_matrix.h.

14.68.3.4 testPixel() `bool QwtPixelMatrix::testPixel ( int x, int y ) const [inline]`

Test if a pixel has been set.

Parameters

x	X-coordinate
y	Y-coordinate

Returns

true, when pos is outside of rect(), or when the pixel has already been set.

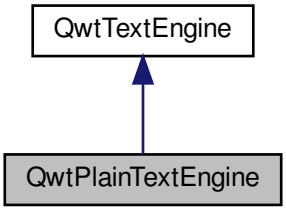
Definition at line 51 of file qwt_pixel_matrix.h.

14.69 QwtPlainTextEngine Class Reference

A text engine for plain texts.

```
#include <qwt_text_engine.h>
```

Inheritance diagram for QwtPlainTextEngine:



Public Member Functions

- QwtPlainTextEngine ()

  Constructor.
- virtual ~QwtPlainTextEngine ()

  Destructor.
- virtual double heightForWidth (const QFont &font, int flags, const QString &text, double width) const override
- virtual QSizeF textSize (const QFont &font, int flags, const QString &text) const override
- virtual void draw (QPainter *, const QRectF &rect, int flags, const QString &text) const override

  Draw the text in a clipping rectangle.
- virtual bool mightRender (const QString &) const override
- virtual void textMargins (const QFont &, const QString &, double &left, double &right, double &top, double &bottom) const override

Additional Inherited Members

14.69.1 Detailed Description

A text engine for plain texts.

QwtPlainTextEngine renders texts using the basic Qt classes QPainter and QFontMetrics.

Definition at line 111 of file qwt_text_engine.h.

14.69.2 Member Function Documentation

14.69.2.1 draw()

```
void QwtPlainTextEngine::draw (
    QPainter * painter,
    const QRectF & rect,
    int flags,
    const QString & text ) const  [override], [virtual]
```

Draw the text in a clipping rectangle.

A wrapper for QPainter::drawText.

Parameters

painter	Painter
rect	Clipping rectangle
flags	Bitwise OR of the flags used like in QPainter::drawText
text	Text to be rendered

Implements QwtTextEngine.

Definition at line 230 of file qwt_text_engine.cpp.

14.69.2.2 heightForWidth()

```
double QwtPlainTextEngine::heightForWidth (
    const QFont & font,
    int flags,
    const QString & text,
    double width ) const  [override], [virtual]
```

Find the height for a given width

Parameters

font	Font of the text
flags	Bitwise OR of the flags used like in QPainter::drawText
text	Text to be rendered
width	Width

Returns

Calculated height

Implements QwtTextEngine.

Definition at line 172 of file qwt_text_engine.cpp.

14.69.2.3 mightRender()

```
bool QwtPlainTextEngine::mightRender (
    const QString &  ) const  [override], [virtual]
```

Test if a string can be rendered by this text engine.

Returns

Always true. All texts can be rendered by QwtPlainTextEngine

Implements QwtTextEngine.

Definition at line 240 of file qwt_text_engine.cpp.

14.69.2.4 textMargins()

```
void QwtPlainTextEngine::textMargins (
    const QFont & font,
    const QString & ,
    double & left,
    double & right,
    double & top,
    double & bottom ) const  [override], [virtual]
```

Return margins around the texts

Parameters

font	Font of the text
left	Return 0
right	Return 0
top	Return value for the top margin
bottom	Return value for the bottom margin

Implements QwtTextEngine.

Definition at line 210 of file qwt_text_engine.cpp.

14.69.2.5 textSize()

```
QSizeF QwtPlainTextEngine::textSize (
    const QFont & font,
```

```
int flags,
const QString & text ) const  [override], [virtual]
```

Returns the size, that is needed to render text

Parameters

font	Font of the text
flags	Bitwise OR of the flags used like in QPainter::drawText
text	Text to be rendered

Returns

Calculated size

Implements QwtTextEngine.

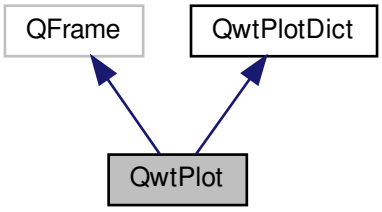
Definition at line 191 of file qwt_text_engine.cpp.

14.70 QwtPlot Class Reference

A 2-D plotting widget.

`#include <qwt_plot.h>`

Inheritance diagram for QwtPlot:



Public Types

- enum LegendPosition { LeftLegend , RightLegend , BottomLegend , TopLegend }
- enum **Axis** {
  yLeft = QwtAxis::YLeft , **yRight** = QwtAxis::YRight , **xBottom** = QwtAxis::XBottom , **xTop** = QwtAxis::XTop ,
  axisCnt = QwtAxis::AxisPositions }

Public Slots

- virtual void replot ()
  Redraw the plot.
- void autoRefresh ()
  Replots the plot if autoReplot() is `true`.

Signals

- void itemAttached (QwtPlotItem *plotItem, bool on)
- void legendDataChanged (const QVariant &itemInfo, const QList< QwtLegendData > &data)

Public Member Functions

- QwtPlot (QWidget *=NULL)

  Constructor.
- QwtPlot (const QwtText &title, QWidget *=NULL)

  Constructor.
- virtual ~QwtPlot ()

  Destructor.
- void setAutoReplot (bool=true)

  Set or reset the autoReplot option.
- bool autoReplot () const
- void setPlotLayout (QwtPlotLayout *)

  Assign a new plot layout.
- QwtPlotLayout * plotLayout ()
- const QwtPlotLayout * plotLayout () const
- void setTitle (const QString &)
- void setTitle (const QwtText &)
- QwtText title () const
- QwtTextLabel * titleLabel ()
- const QwtTextLabel * titleLabel () const
- void setFooter (const QString &)
- void setFooter (const QwtText &)
- QwtText footer () const
- QwtTextLabel * footerLabel ()
- const QwtTextLabel * footerLabel () const
- void setCanvas (QWidget *)

  Set the drawing canvas of the plot widget.
- QWidget * canvas ()
- const QWidget * canvas () const
- void setCanvasBackground (const QBrush &)

  Change the background of the plotting area.
- QBrush canvasBackground () const
- virtual QwtScaleMap canvasMap (QwtAxisId) const
- double invTransform (QwtAxisId, double pos) const
- double transform (QwtAxisId, double value) const

  Transform a value into a coordinate in the plotting region.
- bool isAxisValid (QwtAxisId) const
- void setAxisVisible (QwtAxisId, bool on=true)

  Hide or show a specified axis.
- bool isAxisVisible (QwtAxisId) const
- QwtScaleEngine * axisScaleEngine (QwtAxisId)
- const QwtScaleEngine * axisScaleEngine (QwtAxisId) const
- void setAxisScaleEngine (QwtAxisId, QwtScaleEngine *)
- void setAxisAutoScale (QwtAxisId, bool on=true)

  Enable autoscaling for a specified axis.
- bool axisAutoScale (QwtAxisId) const
- void setAxisFont (QwtAxisId, const QFont &)

Change the font of an axis.

- QFont axisFont (QwtAxisId) const
- void setAxisScale (QwtAxisId, double min, double max, double stepSize=0)

  Disable autoscaling and specify a fixed scale for a selected axis.
- void setAxisScaleDiv (QwtAxisId, const QwtScaleDiv &)

  Disable autoscaling and specify a fixed scale for a selected axis.
- void setAxisScaleDraw (QwtAxisId, QwtScaleDraw ∗)

  Set a scale draw.
- double axisStepSize (QwtAxisId) const

  Return the step size parameter that has been set in setAxisScale.
- QwtInterval axisInterval (QwtAxisId) const

  Return the current interval of the specified axis.
- const QwtScaleDiv & axisScaleDiv (QwtAxisId) const

  Return the scale division of a specified axis.
- const QwtScaleDraw ∗ axisScaleDraw (QwtAxisId) const

  Return the scale draw of a specified axis.
- QwtScaleDraw ∗ axisScaleDraw (QwtAxisId)

  Return the scale draw of a specified axis.
- const QwtScaleWidget ∗ axisWidget (QwtAxisId) const
- QwtScaleWidget ∗ axisWidget (QwtAxisId)
- void setAxisLabelAlignment (QwtAxisId, Qt::Alignment)
- void setAxisLabelRotation (QwtAxisId, double rotation)
- void setAxisTitle (QwtAxisId, const QString &)

  Change the title of a specified axis.
- void setAxisTitle (QwtAxisId, const QwtText &)

  Change the title of a specified axis.
- QwtText axisTitle (QwtAxisId) const
- void setAxisMaxMinor (QwtAxisId, int maxMinor)
- int axisMaxMinor (QwtAxisId) const
- void setAxisMaxMajor (QwtAxisId, int maxMajor)
- int axisMaxMajor (QwtAxisId) const
- void insertLegend (QwtAbstractLegend ∗, LegendPosition=QwtPlot::RightLegend, double ratio=-1.0)

  Insert a legend.
- QwtAbstractLegend ∗ legend ()
- const QwtAbstractLegend ∗ legend () const
- void updateLegend ()
- void updateLegend (const QwtPlotItem ∗)
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override

  Return a minimum size hint.
- virtual void updateLayout ()

  Adjust plot content to its current size.
- virtual void drawCanvas (QPainter ∗)
- void updateAxes ()

  Rebuild the axes scales.
- void updateCanvasMargins ()

  Update the canvas margins.
- virtual void getCanvasMarginsHint (const QwtScaleMap maps[], const QRectF &canvasRect, double &left, double &top, double &right, double &bottom) const

  Calculate the canvas margins.
- virtual bool event (QEvent ∗) override

  Adds handling of layout requests.

- virtual bool eventFilter (QObject *, QEvent *) override

  Event filter.
- virtual void drawItems (QPainter *, const QRectF &, const QwtScaleMap maps[QwtAxis::AxisPositions]) const
- virtual QVariant itemToInfo (QwtPlotItem *) const

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

  Identify the plot item according to an item info object, that has bee generated from itemToInfo().
- void **enableAxis** (int axisId, bool on=true)
- bool **axisEnabled** (int axisId) const

Protected Member Functions

- virtual void resizeEvent (QResizeEvent *) override

14.70.1 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.

The simpleplot example is a good starting point to see how to set up a plot widget.

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" );
     // connect or copy the data to the curves
     curve1->setData( ... );
     curve2->setData( ... );
     curve1->attach( myPlot );
     curve2->attach( myPlot );
     // finally, refresh the plot
     myPlot->replot();
```

Definition at line 78 of file qwt_plot.h.

14.70.2 Member Enumeration Documentation

14.70.2.1 LegendPosition enum QwtPlot::LegendPosition

Position of the legend, relative to the canvas.

See also

insertLegend()

Enumerator

LeftLegend	The legend will be left from the QwtAxis::YLeft axis.
RightLegend	The legend will be right from the QwtAxis::YRight axis.
BottomLegend	The legend will be below the footer.
TopLegend	The legend will be above the title.

Definition at line 93 of file qwt_plot.h.

14.70.3 Constructor & Destructor Documentation

14.70.3.1 QwtPlot() [1/2]

```
QwtPlot::QwtPlot (
    QWidget * parent = NULL )  [explicit]
```

Constructor.

Parameters

parent	Parent widget

Definition at line 103 of file qwt_plot.cpp.

14.70.3.2 QwtPlot() [2/2]

```
QwtPlot::QwtPlot (
    const QwtText & title,
    QWidget * parent = NULL )  [explicit]
```

Constructor.

Parameters

title	Title text
parent	Parent widget

Definition at line 114 of file qwt_plot.cpp.

14.70.4 Member Function Documentation

14.70.4.1 autoReplot()

```
bool QwtPlot::autoReplot ( ) const
```

Returns

true if the autoReplot option is set.

See also

setAutoReplot()

Definition at line 319 of file qwt_plot.cpp.

14.70.4.2 axisAutoScale() `bool QwtPlot::axisAutoScale ( QwtAxisId axisId ) const`

Returns

`True`, if autoscaling is enabled

Parameters

axisId	Axis

Definition at line 213 of file qwt_plot_axis.cpp.

14.70.4.3 axisFont() `QFont QwtPlot::axisFont ( QwtAxisId axisId ) const`

Returns

The font of the scale labels for a specified axis

Parameters

axisId	Axis

Definition at line 237 of file qwt_plot_axis.cpp.

14.70.4.4 axisInterval() `QwtInterval QwtPlot::axisInterval ( QwtAxisId axisId ) const`

Return the current interval of the specified axis.

This is only a convenience function for axisScaleDiv(axisId)->interval();

Parameters

axis↩ Id	Axis

Returns

Scale interval

See also

QwtScaleDiv, axisScaleDiv()

Definition at line 344 of file qwt_plot_axis.cpp.

14.70.4.5 axisMaxMajor() `int QwtPlot::axisMaxMajor ( QwtAxisId axisId ) const`

Returns

The maximum number of major ticks for a specified axis

Parameters

axis↩ Id	Axis

See also

setAxisMaxMajor(), QwtScaleEngine::divideScale()

Definition at line 251 of file qwt_plot_axis.cpp.

14.70.4.6 axisMaxMinor() `int QwtPlot::axisMaxMinor ( QwtAxisId axisId ) const`

Returns

the maximum number of minor ticks for a specified axis

Parameters

axis↩ Id	Axis

See also

setAxisMaxMinor(), QwtScaleEngine::divideScale()

Definition at line 264 of file qwt_plot_axis.cpp.

14.70.4.7 axisScaleDiv() `const QwtScaleDiv & QwtPlot::axisScaleDiv ( QwtAxisId 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

axisId	Axis

Returns

Scale division

See also

QwtScaleDiv, setAxisScaleDiv(), QwtScaleEngine::divideScale()

Definition at line 283 of file qwt_plot_axis.cpp.

14.70.4.8 axisScaleDraw() [1/2] `QwtScaleDraw * QwtPlot::axisScaleDraw ( QwtAxisId axisId )`

Return the scale draw of a specified axis.

Parameters

axisId	Axis

Returns

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

Definition at line 308 of file qwt_plot_axis.cpp.

14.70.4.9 axisScaleDraw() [2/2] const QwtScaleDraw * QwtPlot::axisScaleDraw (QwtAxisId *axisId*) const

Return the scale draw of a specified axis.

Parameters

axisId	Axis

Returns

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

Definition at line 294 of file qwt_plot_axis.cpp.

14.70.4.10 axisScaleEngine() [1/2] QwtScaleEngine * QwtPlot::axisScaleEngine (QwtAxisId *axisId*)

Parameters

axisId	Axis

Returns

Scale engine for a specific axis

Definition at line 190 of file qwt_plot_axis.cpp.

14.70.4.11 axisScaleEngine() [2/2] const QwtScaleEngine * QwtPlot::axisScaleEngine (QwtAxisId *axisId*) const

Parameters

axisId	Axis

Returns

Scale engine for a specific axis

Definition at line 202 of file qwt_plot_axis.cpp.

14.70.4.12 axisStepSize() `double QwtPlot::axisStepSize ( QwtAxisId 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

axisId	Axis

Returns

step size parameter value

See also

setAxisScale(), QwtScaleEngine::divideScale()

Definition at line 326 of file qwt_plot_axis.cpp.

14.70.4.13 axisTitle() `QwtText QwtPlot::axisTitle ( QwtAxisId axisId ) const`

Returns

Title of a specified axis

Parameters

axisId	Axis

Definition at line 356 of file qwt_plot_axis.cpp.

14.70.4.14 axisWidget() [1/2] `QwtScaleWidget * QwtPlot::axisWidget ( QwtAxisId axisId )`

Returns

Scale widget of the specified axis, or NULL if axisId is invalid.

Parameters

axis↩ Id	Axis

Definition at line 153 of file qwt_plot_axis.cpp.

14.70.4.15 axisWidget() [2/2]

```
const QwtScaleWidget * QwtPlot::axisWidget ( QwtAxisId axisId ) const
```

Returns

Scale widget of the specified axis, or NULL if axisId is invalid.

Parameters

axis↩ Id	Axis

Definition at line 141 of file qwt_plot_axis.cpp.

14.70.4.16 canvas() [1/2]

```
QWidget * QwtPlot::canvas ( )
```

Returns

the plot's canvas

Definition at line 463 of file qwt_plot.cpp.

14.70.4.17 canvas() [2/2]

```
const QWidget * QwtPlot::canvas ( ) const
```

Returns

the plot's canvas

Definition at line 471 of file qwt_plot.cpp.

14.70.4.18 canvasBackground() `QBrush QwtPlot::canvasBackground ( ) const`

Nothing else than: canvas()->palette().brush(QPalette::Normal, QPalette::Window);

Returns

Background brush of the plotting area.

See also

setCanvasBackground()

Definition at line 888 of file qwt_plot.cpp.

14.70.4.19 canvasMap() `QwtScaleMap QwtPlot::canvasMap ( QwtAxisId axisId ) const [virtual]`

Parameters

axisId	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()

Definition at line 800 of file qwt_plot.cpp.

14.70.4.20 drawCanvas() `void QwtPlot::drawCanvas ( QPainter * painter ) [virtual]`

Redraw the canvas.

Parameters

painter	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()

Definition at line 742 of file qwt_plot.cpp.

14.70.4.21 drawItems()

```
void QwtPlot::drawItems (
    QPainter * painter,
    const QRectF & canvasRect,
    const QwtScaleMap maps[QwtAxis::AxisPositions] ) const  [virtual]
```

Redraw the canvas items.

Parameters

painter	Painter used for drawing
canvasRect	Bounding rectangle where to paint
maps	QwtAxis::AxisCount maps, mapping between plot and paint device coordinates

Note

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

Definition at line 764 of file qwt_plot.cpp.

14.70.4.22 event()

```
bool QwtPlot::event (
    QEvent * event )  [override], [virtual]
```

Adds handling of layout requests.

Parameters

event	Event

Returns

See QFrame::event()

Definition at line 237 of file qwt_plot.cpp.

14.70.4.23 eventFilter() `bool QwtPlot::eventFilter ( QObject * object, QEvent * event ) [override], [virtual]`

Event filter.

The plot handles the following events for the canvas:

- QEvent::Resize The canvas margins might depend on its size
- QEvent::ContentsRectChange The layout needs to be recalculated

Parameters

object	Object to be filtered
event	Event

Returns

See QFrame::eventFilter()

See also

updateCanvasMargins(), updateLayout()

Definition at line 271 of file qwt_plot.cpp.

14.70.4.24 footer() `QwtText QwtPlot::footer ( ) const`

Returns

Text of the footer

Definition at line 395 of file qwt_plot.cpp.

14.70.4.25 footerLabel() [1/2] `QwtTextLabel * QwtPlot::footerLabel ( )`

Returns

Footer label widget.

Definition at line 401 of file qwt_plot.cpp.

14.70.4.26 footerLabel() [2/2] const QwtTextLabel * QwtPlot::footerLabel ( ) const

Returns

Footer label widget.

Definition at line 407 of file qwt_plot.cpp.

14.70.4.27 getCanvasMarginsHint()

```
void QwtPlot::getCanvasMarginsHint (
    const QwtScaleMap maps[],
    const QRectF & canvasRect,
    double & left,
    double & top,
    double & right,
    double & bottom ) const  [virtual]
```

Calculate the canvas margins.

Parameters

maps	QwtAxis::AxisCount maps, mapping between plot and paint device coordinates
canvasRect	Bounding rectangle where to paint
left	Return parameter for the left margin
top	Return parameter for the top margin
right	Return parameter for the right margin
bottom	Return parameter for the bottom margin

Plot items might indicate, that they need some extra space at the borders of the canvas by the QwtPlotItem::Margins flag.

updateCanvasMargins(), QwtPlotItem::getCanvasMarginHint()

Definition at line 670 of file qwt_plot.cpp.

14.70.4.28 infoToItem()

```
QwtPlotItem * QwtPlot::infoToItem (
    const QVariant & itemInfo ) const  [virtual]
```

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

The default implementation simply tries to unwrap a QwtPlotItem pointer:

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

Parameters

itemInfo	Plot item

Returns

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

See also

itemToInfo()

Definition at line 1159 of file qwt_plot.cpp.

14.70.4.29 insertLegend()

```
void QwtPlot::insertLegend (
    QwtAbstractLegend * 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.

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

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

Parameters

legend	Legend
pos	The legend's position. For top/left position the number of columns will be limited to 1, otherwise it will be set to unlimited.
ratio	Ratio between legend and the bounding rectangle 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.

See also

legend(), QwtPlotLayout::legendPosition(), QwtPlotLayout::setLegendPosition()

Definition at line 927 of file qwt_plot.cpp.

14.70.4.30 invTransform()

```
double QwtPlot::invTransform (
    QwtAxisId axisId,
    double pos ) const
```

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

Parameters

axis↩ Id	Axis
pos	position

Returns

Position as axis coordinate

Warning

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

Definition at line 397 of file qwt_plot_axis.cpp.

14.70.4.31 isAxisValid() `bool QwtPlot::isAxisValid ( QwtAxisId axisId ) const`

Checks if an axis is valid

Parameters

axis↩ Id	axis

Returns

`true` if the specified axis exists, otherwise `false`

Note

This method is equivalent to QwtAxis::isValid(axisId) and simply checks if axisId is one of the values of QwtAxis::Position. It is a placeholder for future releases, where it will be possible to have a customizable number of axes (multiaxes branch) at each side.

Definition at line 132 of file qwt_plot_axis.cpp.

14.70.4.32 isAxisVisible() `bool QwtPlot::isAxisVisible ( QwtAxisId axisId ) const`

Returns

`True`, if a specified axis is visible

Parameters

axis↩ Id	Axis

Definition at line 225 of file qwt_plot_axis.cpp.

14.70.4.33 itemAttached

```
void QwtPlot::itemAttached (
    QwtPlotItem * plotItem,
    bool on )  [signal]
```

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

Parameters

plotItem	Plot item
on	Attached/Detached

14.70.4.34 itemToInfo()

```
QVariant QwtPlot::itemToInfo (
    QwtPlotItem * plotItem ) const  [virtual]
```

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

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

Parameters

plotItem	Plot item

Returns

Plot item embedded in a QVariant

See also

infoToItem()

Definition at line 1139 of file qwt_plot.cpp.

14.70.4.35 legend() [1/2] QwtAbstractLegend * QwtPlot::legend ( )

Returns

the plot's legend

See also

insertLegend()

Definition at line 445 of file qwt_plot.cpp.

14.70.4.36 legend() [2/2] const QwtAbstractLegend * QwtPlot::legend ( ) const

Returns

the plot's legend

See also

insertLegend()

Definition at line 454 of file qwt_plot.cpp.

14.70.4.37 legendDataChanged

```
void QwtPlot::legendDataChanged (
    const QVariant & itemInfo,
    const QList< QwtLegendData > & data ) [signal]
```

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

Parameters

itemInfo	Info about a plot item, build from itemToInfo()
data	Attributes of the entries (usually <= 1) for the plot item.

See also

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

14.70.4.38 plotLayout() [1/2] QwtPlotLayout * QwtPlot::plotLayout ( )

Returns

the plot's layout

Definition at line 430 of file qwt_plot.cpp.

14.70.4.39 plotLayout() [2/2] const QwtPlotLayout * QwtPlot::plotLayout ( ) const

Returns

the plot's layout

Definition at line 436 of file qwt_plot.cpp.

14.70.4.40 replot 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

updateAxes(), setAutoReplot()

Definition at line 545 of file qwt_plot.cpp.

14.70.4.41 resizeEvent() void QwtPlot::resizeEvent (QResizeEvent * e) [override], [protected], [virtual]

Resize and update internal layout

Parameters

e	Resize event

Definition at line 530 of file qwt_plot.cpp.

14.70.4.42 setAutoReplot() 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

tf	`true` or `false`. Defaults to `true`.

See also

replot()

Definition at line 310 of file qwt_plot.cpp.

14.70.4.43 setAxisAutoScale() `void QwtPlot::setAxisAutoScale (`

```
QwtAxisId axisId,
bool on = true )
```

Enable autoscaling for a specified axis.

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

Parameters

axisId	Axis
on	On/Off

See also

setAxisScale(), setAxisScaleDiv(), updateAxes()

Note

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

Definition at line 449 of file qwt_plot_axis.cpp.

14.70.4.44 setAxisFont() `void QwtPlot::setAxisFont (`

```
QwtAxisId axisId,
const QFont & font )
```

Change the font of an axis.

Parameters

axis↩ Id	Axis
font	Font

Warning

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

Definition at line 430 of file qwt_plot_axis.cpp.

14.70.4.45 setAxisLabelAlignment() `void QwtPlot::setAxisLabelAlignment ( QwtAxisId axisId, Qt::Alignment alignment )`

Change the alignment of the tick labels

Parameters

axisId	Axis
alignment	Or'd Qt::AlignmentFlags see <qnamespace.h>

See also

QwtScaleDraw::setLabelAlignment()

Definition at line 549 of file qwt_plot_axis.cpp.

14.70.4.46 setAxisLabelRotation() `void QwtPlot::setAxisLabelRotation ( QwtAxisId axisId, double rotation )`

Rotate all tick labels

Parameters

axisId	Axis
rotation	Angle in degrees. When changing the label rotation, the label alignment might be adjusted too.

See also

QwtScaleDraw::setLabelRotation(), setAxisLabelAlignment()

Definition at line 564 of file qwt_plot_axis.cpp.

14.70.4.47 setAxisMaxMajor()

```
void QwtPlot::setAxisMaxMajor (
    QwtAxisId axisId,
    int maxMajor )
```

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

Parameters

axisId	Axis
maxMajor	Maximum number of major steps

See also

axisMaxMajor()

Definition at line 602 of file qwt_plot_axis.cpp.

14.70.4.48 setAxisMaxMinor()

```
void QwtPlot::setAxisMaxMinor (
    QwtAxisId axisId,
    int maxMinor )
```

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

Parameters

axisId	Axis
maxMinor	Maximum number of minor steps

See also

axisMaxMinor()

Definition at line 578 of file qwt_plot_axis.cpp.

14.70.4.49 setAxisScale()

```
void QwtPlot::setAxisScale (
    QwtAxisId axisId,
    double min,
    double max,
    double stepSize = 0 )
```

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

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

Parameters

axisId	Axis
min	Minimum of the scale
max	Maximum of the scale
stepSize	Major step size. If `step == 0`, the step size is calculated automatically using the maxMajor setting.

See also

setAxisMaxMajor(), setAxisAutoScale(), axisStepSize(), QwtScaleEngine::divideScale()

Definition at line 473 of file qwt_plot_axis.cpp.

14.70.4.50 setAxisScaleDiv()

```
void QwtPlot::setAxisScaleDiv (
    QwtAxisId axisId,
    const QwtScaleDiv & scaleDiv )
```

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

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

Parameters

axisId	Axis
scaleDiv	Scale division

See also

setAxisScale(), setAxisAutoScale()

Definition at line 502 of file qwt_plot_axis.cpp.

14.70.4.51 setAxisScaleDraw()

```
void QwtPlot::setAxisScaleDraw (
    QwtAxisId axisId,
    QwtScaleDraw * scaleDraw )
```

Set a scale draw.

Parameters

axisId	Axis
scaleDraw	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.

Definition at line 532 of file qwt_plot_axis.cpp.

14.70.4.52 setAxisScaleEngine()

```
void QwtPlot::setAxisScaleEngine (
    QwtAxisId axisId,
    QwtScaleEngine * scaleEngine )
```

Change the scale engine for an axis

Parameters

axisId	Axis
scaleEngine	Scale engine

See also

axisScaleEngine()

Definition at line 169 of file qwt_plot_axis.cpp.

14.70.4.53 setAxisTitle() [1/2]

```
void QwtPlot::setAxisTitle (
    QwtAxisId axisId,
    const QString & title )
```

Change the title of a specified axis.

Parameters

axisId	Axis
title	axis title

Definition at line 624 of file qwt_plot_axis.cpp.

14.70.4.54 setAxisTitle() [2/2] void QwtPlot::setAxisTitle (QwtAxisId *axisId*, const QwtText & *title*)

Change the title of a specified axis.

Parameters

axisId	Axis
title	Axis title

Definition at line 636 of file qwt_plot_axis.cpp.

14.70.4.55 setAxisVisible() void QwtPlot::setAxisVisible (QwtAxisId *axisId*, bool *on* = *true*)

Hide or show a specified axis.

Curves, markers and other items can be attached to hidden axes, and transformation of screen coordinates into values works as normal.

Only QwtAxis::XBottom and QwtAxis::YLeft are enabled by default.

Parameters

axisId	Axis
on	`true` (visible) or `false` (hidden)

Definition at line 376 of file qwt_plot_axis.cpp.

14.70.4.56 setCanvas() void QwtPlot::setCanvas (QWidget * *canvas*)

Set the drawing canvas of the plot widget.

QwtPlot invokes methods of the canvas as meta methods (see QMetaObject). In opposite to using conventional C++ techniques like virtual methods they allow to use canvas implementations that are derived from QWidget or QGLWidget.

The following meta methods could be implemented:

- replot() When the canvas doesn't offer a replot method, QwtPlot calls update() instead.
- borderPath() The border path is necessary to clip the content of the canvas When the canvas doesn't have any special border (f.e rounded corners) it is o.k. not to implement this method.

The default canvas is a QwtPlotCanvas

Parameters

canvas	Canvas Widget

See also

canvas()

Definition at line 213 of file qwt_plot.cpp.

14.70.4.57 setCanvasBackground() `void QwtPlot::setCanvasBackground ( const QBrush & brush )`

Change the background of the plotting area.

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

Parameters

brush	New background brush

See also

canvasBackground()

Definition at line 873 of file qwt_plot.cpp.

14.70.4.58 setFooter() [1/2] `void QwtPlot::setFooter ( const QString & text )`

Change the text the footer

Parameters

text	New text of the footer

Definition at line 372 of file qwt_plot.cpp.

14.70.4.59 setFooter() [2/2] `void QwtPlot::setFooter ( const QwtText & text )`

Change the text the footer

Parameters

text	New text of the footer

Definition at line 385 of file qwt_plot.cpp.

14.70.4.60 setPlotLayout() `void QwtPlot::setPlotLayout ( QwtPlotLayout * layout )`

Assign a new plot layout.

Parameters

layout	Layout()

See also

plotLayout()

Definition at line 418 of file qwt_plot.cpp.

14.70.4.61 setTitle() [1/2] `void QwtPlot::setTitle ( const QString & title )`

Change the plot's title

Parameters

title	New title

Definition at line 328 of file qwt_plot.cpp.

14.70.4.62 setTitle() [2/2] `void QwtPlot::setTitle ( const QwtText & title )`

Change the plot's title

Parameters

title	New title

Definition at line 341 of file qwt_plot.cpp.

14.70.4.63 sizeHint() `QSize QwtPlot::sizeHint ( ) const [override], [virtual]`

Returns

Size hint for the plot widget

See also

minimumSizeHint()

Definition at line 480 of file qwt_plot.cpp.

14.70.4.64 title() `QwtText QwtPlot::title ( ) const`

Returns

Title of the plot

Definition at line 351 of file qwt_plot.cpp.

14.70.4.65 titleLabel() [1/2] `QwtTextLabel * QwtPlot::titleLabel ( )`

Returns

Title label widget.

Definition at line 357 of file qwt_plot.cpp.

14.70.4.66 titleLabel() [2/2] `const QwtTextLabel * QwtPlot::titleLabel ( ) const`

Returns

Title label widget.

Definition at line 363 of file qwt_plot.cpp.

14.70.4.67 transform()

```
double QwtPlot::transform (
    QwtAxisId axisId,
    double value ) const
```

Transform a value into a coordinate in the plotting region.

Parameters

axis↩ Id	Axis
value	value

Returns

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

Definition at line 414 of file qwt_plot_axis.cpp.

14.70.4.68 updateAxes() `void QwtPlot::updateAxes ( )`

Rebuild the axes scales.

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

When the scale boundaries have been assigned with setAxisScale() a scale division is calculated (QwtScale↩ Engine::didvideScale()) for this interval and assigned to the scale widget.

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

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

updateAxes() is usually called by replot().

See also

setAxisAutoScale(), setAxisScale(), setAxisScaleDiv(), replot() QwtPlotItem::boundingRect()

Definition at line 666 of file qwt_plot_axis.cpp.

14.70.4.69 updateCanvasMargins() `void QwtPlot::updateCanvasMargins ( )`

Update the canvas margins.

Plot items might indicate, that they need some extra space at the borders of the canvas by the QwtPlotItem::Margins flag.

getCanvasMarginsHint(), QwtPlotItem::getCanvasMarginHint()

Definition at line 706 of file qwt_plot.cpp.

14.70.4.70 updateLayout() void QwtPlot::updateLayout ( ) [virtual]

Adjust plot content to its current size.

See also

resizeEvent()

Definition at line 577 of file qwt_plot.cpp.

14.70.4.71 updateLegend() [1/2] void QwtPlot::updateLegend ( )

Emit legendDataChanged() for all plot item

See also

QwtPlotItem::legendData(), legendDataChanged()

Definition at line 1016 of file qwt_plot.cpp.

14.70.4.72 updateLegend() [2/2] void QwtPlot::updateLegend (const QwtPlotItem * *plotItem*)

Emit legendDataChanged() for a plot item

Parameters

plotItem	Plot item

See also

QwtPlotItem::legendData(), legendDataChanged()

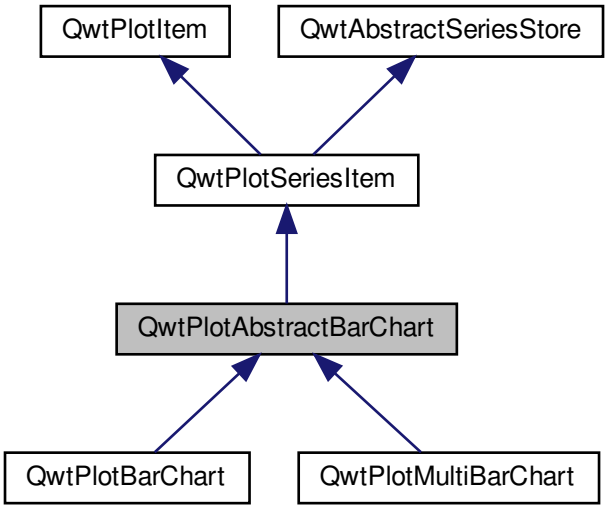
Definition at line 1032 of file qwt_plot.cpp.

14.71 QwtPlotAbstractBarChart Class Reference

Abstract base class for bar chart items.

`#include <qwt_plot_abstract_barchart.h>`

Inheritance diagram for QwtPlotAbstractBarChart:



Public Types

- enum LayoutPolicy { AutoAdjustSamples , ScaleSamplesToAxes , ScaleSampleToCanvas , FixedSampleSize }

  Mode how to calculate the bar width.

Public Member Functions

- QwtPlotAbstractBarChart (const QwtText &title)
- virtual ~QwtPlotAbstractBarChart ()

  Destructor.
- void setLayoutPolicy (LayoutPolicy)
- LayoutPolicy layoutPolicy () const
- void setLayoutHint (double)
- double layoutHint () const
- void setSpacing (int)

  Set the spacing.
- int spacing () const
- void setMargin (int)

  Set the margin.
- int margin () const
- void setBaseline (double)

  Set the baseline.
- double baseline () const
- virtual void getCanvasMarginHint (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, double &left, double &top, double &right, double &bottom) const override

  Calculate a hint for the canvas margin.

Protected Member Functions

- double sampleWidth (const QwtScaleMap &map, double canvasSize, double boundingSize, double value) const

14.71.1 Detailed Description

Abstract base class for bar chart items.

In opposite to almost all other plot items bar charts can't be displayed inside of their bounding rectangle and need a special API how to calculate the width of the bars and how they affect the layout of the attached plot.

Definition at line 24 of file qwt_plot_abstract_barchart.h.

14.71.2 Member Enumeration Documentation

14.71.2.1 LayoutPolicy `enum QwtPlotAbstractBarChart::LayoutPolicy`

Mode how to calculate the bar width.

setLayoutPolicy(), setLayoutHint(), barWidthHint()

Enumerator

AutoAdjustSamples	The sample width is calculated by dividing the bounding rectangle by the number of samples. The layoutHint() is used as a minimum width in paint device coordinates. **See also** boundingRectangle()
ScaleSamplesToAxes	layoutHint() defines an interval in axis coordinates
ScaleSampleToCanvas	The bar width is calculated by multiplying layoutHint() with the height or width of the canvas. **See also** boundingRectangle()
FixedSampleSize	layoutHint() defines a fixed width in paint device coordinates.

Definition at line 32 of file qwt_plot_abstract_barchart.h.

14.71.3 Constructor & Destructor Documentation

14.71.3.1 QwtPlotAbstractBarChart() `QwtPlotAbstractBarChart::QwtPlotAbstractBarChart ( const QwtText & title ) [explicit]`

Constructor

Parameters

title	Title of the chart

Definition at line 48 of file qwt_plot_abstract_barchart.cpp.

14.71.4 Member Function Documentation

14.71.4.1 baseline() `double QwtPlotAbstractBarChart::baseline ( ) const`

Returns

Value for the origin of the bar chart

See also

setBaseline(), QwtPlotSeriesItem::orientation()

Definition at line 208 of file qwt_plot_abstract_barchart.cpp.

14.71.4.2 getCanvasMarginHint() `void QwtPlotAbstractBarChart::getCanvasMarginHint (`

```
const QwtScaleMap & xMap,
const QwtScaleMap & yMap,
const QRectF & canvasRect,
double & left,
double & top,
double & right,
double & bottom ) const  [override], [virtual]
```

Calculate a hint for the canvas margin.

Bar charts need to reserve some space for displaying the bars for the first and the last sample. The hint is calculated from the layoutHint() depending on the layoutPolicy().

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

Parameters

xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas in painter coordinates
left	Returns the left margin
top	Returns the top margin
right	Returns the right margin
bottom	Returns the bottom margin

Returns

Margin

See also

layoutPolicy(), layoutHint(), QwtPlotItem::Margins QwtPlot::getCanvasMarginsHint(), QwtPlot::updateCanvasMargins()

Reimplemented from QwtPlotItem.

Definition at line 289 of file qwt_plot_abstract_barchart.cpp.

14.71.4.3 layoutHint() `double QwtPlotAbstractBarChart::layoutHint ( ) const`

The combination of layoutPolicy() and layoutHint() define how the width of the bars is calculated

Returns

Layout policy of the chart item

See also

LayoutPolicy, setLayoutHint(), layoutPolicy()

Definition at line 119 of file qwt_plot_abstract_barchart.cpp.

14.71.4.4 layoutPolicy() `QwtPlotAbstractBarChart::LayoutPolicy QwtPlotAbstractBarChart::layout↩Policy ( ) const`

The combination of layoutPolicy() and layoutHint() define how the width of the bars is calculated

Returns

Layout policy of the chart item

See also

setLayoutPolicy(), layoutHint()

Definition at line 89 of file qwt_plot_abstract_barchart.cpp.

14.71.4.5 margin() `int QwtPlotAbstractBarChart::margin ( ) const`

Returns

Margin between the outmost bars and the contentsRect() of the canvas.

See also

setMargin(), spacing()

Definition at line 176 of file qwt_plot_abstract_barchart.cpp.

14.71.4.6 sampleWidth() `double QwtPlotAbstractBarChart::sampleWidth (`

```
const QwtScaleMap & map,
double canvasSize,
double boundingSize,
double value ) const  [protected]
```

Calculate the width for a sample in paint device coordinates

Parameters

map	Scale map for the corresponding scale
canvasSize	Size of the canvas in paint device coordinates
boundingSize	Bounding size of the chart in plot coordinates (used in AutoAdjustSamples mode)
value	Value of the sample

Returns

Sample width

See also

layoutPolicy(), layoutHint()

Definition at line 225 of file qwt_plot_abstract_barchart.cpp.

14.71.4.7 setBaseline() `void QwtPlotAbstractBarChart::setBaseline ( double value )`

Set the baseline.

The baseline is the origin for the chart. Each bar is painted from the baseline in the direction of the sample value. In case of a horizontal orientation() the baseline is interpreted as x - otherwise as y - value.

The default value for the baseline is 0.

Parameters

value	Value for the baseline

See also

baseline(), QwtPlotSeriesItem::orientation()

Definition at line 195 of file qwt_plot_abstract_barchart.cpp.

14.71.4.8 setLayoutHint() `void QwtPlotAbstractBarChart::setLayoutHint ( double hint )`

The combination of layoutPolicy() and layoutHint() define how the width of the bars is calculated

Parameters

hint	Layout hint

See also

LayoutPolicy, layoutPolicy(), layoutHint()

Definition at line 102 of file qwt_plot_abstract_barchart.cpp.

14.71.4.9 setLayoutPolicy() `void QwtPlotAbstractBarChart::setLayoutPolicy ( LayoutPolicy policy )`

The combination of layoutPolicy() and layoutHint() define how the width of the bars is calculated

Parameters

policy	Layout policy

See also

layoutPolicy(), layoutHint()

Definition at line 73 of file qwt_plot_abstract_barchart.cpp.

14.71.4.10 setMargin() `void QwtPlotAbstractBarChart::setMargin ( int margin )`

Set the margin.

The margin is the distance between the outmost bars and the contentsRect() of the canvas. The default setting is 5 pixels.

Parameters

margin	Margin

See also

spacing(), margin()

Definition at line 160 of file qwt_plot_abstract_barchart.cpp.

14.71.4.11 setSpacing() `void QwtPlotAbstractBarChart::setSpacing ( int spacing )`

Set the spacing.

The spacing is the distance between 2 samples (bars for QwtPlotBarChart or a group of bars for QwtPlotMultiBarChart) in paint device coordinates.

See also

spacing()

Definition at line 132 of file qwt_plot_abstract_barchart.cpp.

14.71.4.12 spacing() `int QwtPlotAbstractBarChart::spacing ( ) const`

Returns

Spacing between 2 samples (bars or groups of bars)

See also

setSpacing(), margin()

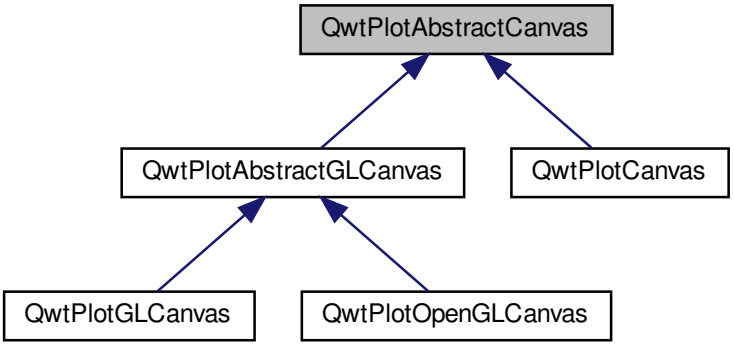
Definition at line 146 of file qwt_plot_abstract_barchart.cpp.

14.72 QwtPlotAbstractCanvas Class Reference

Base class for all type of plot canvases.

`#include <qwt_plot_abstract_canvas.h>`

Inheritance diagram for QwtPlotAbstractCanvas:



Public Types

- enum FocusIndicator { NoFocusIndicator , CanvasFocusIndicator , ItemFocusIndicator }

  Focus indicator The default setting is NoFocusIndicator.

Public Member Functions

- QwtPlotAbstractCanvas (QWidget *canvasWidget)
  Constructor.
- virtual ~QwtPlotAbstractCanvas ()
  Destructor.
- QwtPlot * plot ()
  Return parent plot widget.
- const QwtPlot * plot () const
  Return parent plot widget.
- void setFocusIndicator (FocusIndicator)
- FocusIndicator focusIndicator () const
- void setBorderRadius (double)
- double borderRadius () const

Protected Member Functions

- QWidget * canvasWidget ()
- const QWidget * canvasWidget () const
- virtual void drawFocusIndicator (QPainter *)
- virtual void drawBorder (QPainter *)
- virtual void drawBackground (QPainter *)
  Helper function for the derived plot canvas.
- void fillBackground (QPainter *)
  Helper function for the derived plot canvas.
- void drawCanvas (QPainter *)
  Draw the plot to the canvas.
- void drawStyled (QPainter *, bool)
  Helper function for the derived plot canvas.
- void drawUnstyled (QPainter *)
  Helper function for the derived plot canvas.
- QPainterPath canvasBorderPath (const QRect &rect) const
- void updateStyleSheetInfo ()
  Update the cached information about the current style sheet.

14.72.1 Detailed Description

Base class for all type of plot canvases.

Definition at line 21 of file qwt_plot_abstract_canvas.h.

14.72.2 Member Enumeration Documentation

14.72.2.1 FocusIndicator `enum QwtPlotAbstractCanvas::FocusIndicator`

Focus indicator The default setting is NoFocusIndicator.

See also

setFocusIndicator(), focusIndicator(), drawFocusIndicator()

Enumerator

NoFocusIndicator	Don't paint a focus indicator.
CanvasFocusIndicator	The focus is related to the complete canvas. Paint the focus indicator using drawFocusIndicator()
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.

Definition at line 30 of file qwt_plot_abstract_canvas.h.

14.72.3 Constructor & Destructor Documentation

14.72.3.1 QwtPlotAbstractCanvas() `QwtPlotAbstractCanvas::QwtPlotAbstractCanvas ( QWidget * canvasWidget ) [explicit]`

Constructor.

Parameters

canvasWidget	plot canvas widget

Definition at line 534 of file qwt_plot_abstract_canvas.cpp.

14.72.4 Member Function Documentation

14.72.4.1 borderRadius() `double QwtPlotAbstractCanvas::borderRadius ( ) const`

Returns

Radius for the corners of the border frame

See also

setBorderRadius()

Definition at line 613 of file qwt_plot_abstract_canvas.cpp.

14.72.4.2 canvasBorderPath() QPainterPath QwtPlotAbstractCanvas::canvasBorderPath (const QRect & *rect*) const [protected]

Returns

Path for the canvas border

Definition at line 619 of file qwt_plot_abstract_canvas.cpp.

14.72.4.3 canvasWidget() [1/2] QWidget * QwtPlotAbstractCanvas::canvasWidget ( ) [protected]

Returns

canvas widget

Definition at line 872 of file qwt_plot_abstract_canvas.cpp.

14.72.4.4 canvasWidget() [2/2] const QWidget * QwtPlotAbstractCanvas::canvasWidget ( ) const [protected]

Returns

canvas widget

Definition at line 878 of file qwt_plot_abstract_canvas.cpp.

14.72.4.5 drawBorder() void QwtPlotAbstractCanvas::drawBorder (QPainter * *painter*) [protected], [virtual]

Draw the border of the canvas

Parameters

painter	Painter

Reimplemented in QwtPlotCanvas.

Definition at line 628 of file qwt_plot_abstract_canvas.cpp.

14.72.4.6 drawFocusIndicator() void QwtPlotAbstractCanvas::drawFocusIndicator (QPainter * *painter*) [protected], [virtual]

Draw the focus indication

Parameters

painter	Painter

Definition at line 587 of file qwt_plot_abstract_canvas.cpp.

14.72.4.7 focusIndicator() `QwtPlotAbstractCanvas::FocusIndicator QwtPlotAbstractCanvas::focusIndicator ( ) const`

Returns

Focus indicator

See also

FocusIndicator, setFocusIndicator()

Definition at line 578 of file qwt_plot_abstract_canvas.cpp.

14.72.4.8 setBorderRadius() `void QwtPlotAbstractCanvas::setBorderRadius ( double radius )`

Set the radius for the corners of the border frame

Parameters

radius	Radius of a rounded corner

See also

borderRadius()

Definition at line 604 of file qwt_plot_abstract_canvas.cpp.

14.72.4.9 setFocusIndicator() `void QwtPlotAbstractCanvas::setFocusIndicator ( FocusIndicator focusIndicator )`

Set the focus indicator

See also

FocusIndicator, focusIndicator()

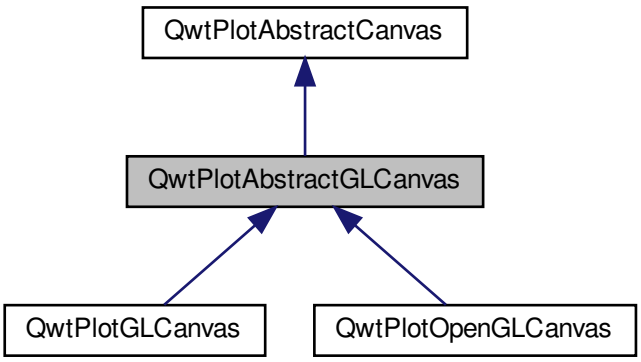
Definition at line 568 of file qwt_plot_abstract_canvas.cpp.

14.73 QwtPlotAbstractGLCanvas Class Reference

Base class of QwtPlotOpenGLCanvas and QwtPlotGLCanvas.

`#include <qwt_plot_abstract_canvas.h>`

Inheritance diagram for QwtPlotAbstractGLCanvas:



Public Types

- enum PaintAttribute { BackingStore = 1 , ImmediatePaint = 8 }
  Paint attributes.
- typedef QFlags< PaintAttribute > PaintAttributes
  Paint attributes.

Public Member Functions

- QwtPlotAbstractGLCanvas (QWidget *canvasWidget)
  Constructor.
- virtual ~QwtPlotAbstractGLCanvas ()
  Destructor.
- void setPaintAttribute (PaintAttribute, bool on=true)
  Changing the paint attributes.
- bool testPaintAttribute (PaintAttribute) const
- void setFrameStyle (int style)
- int frameStyle () const
- void setFrameShadow (QFrame::Shadow)
- QFrame::Shadow frameShadow () const
- void setFrameShape (QFrame::Shape)
- QFrame::Shape frameShape () const
- void setLineWidth (int)
- int lineWidth () const
- void setMidLineWidth (int)
- int midLineWidth () const
- int frameWidth () const
- QRect frameRect () const
- virtual void invalidateBackingStore ()=0
  Invalidate the internal backing store.

Protected Member Functions

- void replot ()
- void draw (QPainter ∗)

  Helper function for the derived plot canvas.

14.73.1 Detailed Description

Base class of QwtPlotOpenGLCanvas and QwtPlotGLCanvas.

Definition at line 87 of file qwt_plot_abstract_canvas.h.

14.73.2 Member Typedef Documentation

14.73.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotAbstractGLCanvas::PaintAttributes`

Paint attributes.

An ORed combination of PaintAttribute values.

Definition at line 122 of file qwt_plot_abstract_canvas.h.

14.73.3 Member Enumeration Documentation

14.73.3.1 PaintAttribute `enum QwtPlotAbstractGLCanvas::PaintAttribute`

Paint attributes.

The default setting enables BackingStore and Opaque.

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

BackingStore	Paint double buffered reusing the content of the pixmap buffer when possible. Using a backing store might improve the performance significantly, when working with widget overlays (like rubber bands). Disabling the cache might improve the performance for incremental paints (using QwtPlotDirectPainter). **See also** backingStore(), invalidateBackingStore()
ImmediatePaint	When ImmediatePaint is set replot() calls repaint() instead of update(). **See also** replot(), QWidget::repaint(), QWidget::update()

Definition at line 97 of file qwt_plot_abstract_canvas.h.

14.73.4 Constructor & Destructor Documentation

14.73.4.1 QwtPlotAbstractGLCanvas() `QwtPlotAbstractGLCanvas::QwtPlotAbstractGLCanvas ( QWidget * canvasWidget ) [explicit]`

Constructor.

Parameters

canvasWidget	plot canvas widget

Definition at line 904 of file qwt_plot_abstract_canvas.cpp.

14.73.5 Member Function Documentation

14.73.5.1 frameRect() `QRect QwtPlotAbstractGLCanvas::frameRect ( ) const`

Returns

The rectangle where the frame is drawn in.

Definition at line 1105 of file qwt_plot_abstract_canvas.cpp.

14.73.5.2 frameShadow() `QFrame::Shadow QwtPlotAbstractGLCanvas::frameShadow ( ) const`

Returns

Frame shadow

See also

setFrameShadow(), QFrame::setFrameShadow()

Definition at line 1000 of file qwt_plot_abstract_canvas.cpp.

14.73.5.3 frameShape() `QFrame::Shape QwtPlotAbstractGLCanvas::frameShape ( ) const`

Returns

Frame shape

See also

setFrameShape(), QFrame::frameShape()

Definition at line 1020 of file qwt_plot_abstract_canvas.cpp.

14.73.5.4 frameStyle() `int QwtPlotAbstractGLCanvas::frameStyle ( ) const`

Returns

The bitwise OR between a frameShape() and a frameShadow()

See also

setFrameStyle(), QFrame::frameStyle()

Definition at line 980 of file qwt_plot_abstract_canvas.cpp.

14.73.5.5 frameWidth() `int QwtPlotAbstractGLCanvas::frameWidth ( ) const`

Returns

Frame width depending on the style, line width and midline width.

Definition at line 1084 of file qwt_plot_abstract_canvas.cpp.

14.73.5.6 lineWidth() `int QwtPlotAbstractGLCanvas::lineWidth ( ) const`

Returns

Line width of the frame

See also

setLineWidth(), midLineWidth()

Definition at line 1048 of file qwt_plot_abstract_canvas.cpp.

14.73.5.7 midLineWidth() `int QwtPlotAbstractGLCanvas::midLineWidth ( ) const`

Returns

Midline width of the frame

See also

setMidLineWidth(), lineWidth()

Definition at line 1076 of file qwt_plot_abstract_canvas.cpp.

14.73.5.8 replot() `void QwtPlotAbstractGLCanvas::replot ( ) [protected]`

Invalidate the paint cache and repaint the canvas

See also

invalidatePaintCache()

Definition at line 1093 of file qwt_plot_abstract_canvas.cpp.

14.73.5.9 setFrameShadow() `void QwtPlotAbstractGLCanvas::setFrameShadow ( QFrame::Shadow shadow )`

Set the frame shadow

Parameters

shadow	Frame shadow

See also

frameShadow(), setFrameShape(), QFrame::setFrameShadow()

Definition at line 991 of file qwt_plot_abstract_canvas.cpp.

14.73.5.10 setFrameShape() `void QwtPlotAbstractGLCanvas::setFrameShape ( QFrame::Shape shape )`

Set the frame shape

Parameters

shape	Frame shape

See also

frameShape(), setFrameShadow(), QFrame::frameShape()

Definition at line 1011 of file qwt_plot_abstract_canvas.cpp.

14.73.5.11 setFrameStyle() `void QwtPlotAbstractGLCanvas::setFrameStyle ( int style )`

Set the frame style

Parameters

style	The bitwise OR between a shape and a shadow.

See also

frameStyle(), QFrame::setFrameStyle(), setFrameShadow(), setFrameShape()

Definition at line 965 of file qwt_plot_abstract_canvas.cpp.

14.73.5.12 setLineWidth() `void QwtPlotAbstractGLCanvas::setLineWidth ( int width )`

Set the frame line width

The default line width is 2 pixels.

Parameters

width	Line width of the frame

See also

lineWidth(), setMidLineWidth()

Definition at line 1033 of file qwt_plot_abstract_canvas.cpp.

14.73.5.13 setMidLineWidth() `void QwtPlotAbstractGLCanvas::setMidLineWidth ( int width )`

Set the frame mid line width

The default midline width is 0 pixels.

Parameters

width	Midline width of the frame

See also

midLineWidth(), setLineWidth()

Definition at line 1061 of file qwt_plot_abstract_canvas.cpp.

14.73.5.14 setPaintAttribute() `void QwtPlotAbstractGLCanvas::setPaintAttribute ( PaintAttribute attribute, bool on = true )`

Changing the paint attributes.

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 927 of file qwt_plot_abstract_canvas.cpp.

14.73.5.15 testPaintAttribute() `bool QwtPlotAbstractGLCanvas::testPaintAttribute ( PaintAttribute attribute ) const`

Test whether a paint attribute is enabled

Parameters

attribute	Paint attribute

Returns

true, when attribute is enabled

See also

setPaintAttribute()

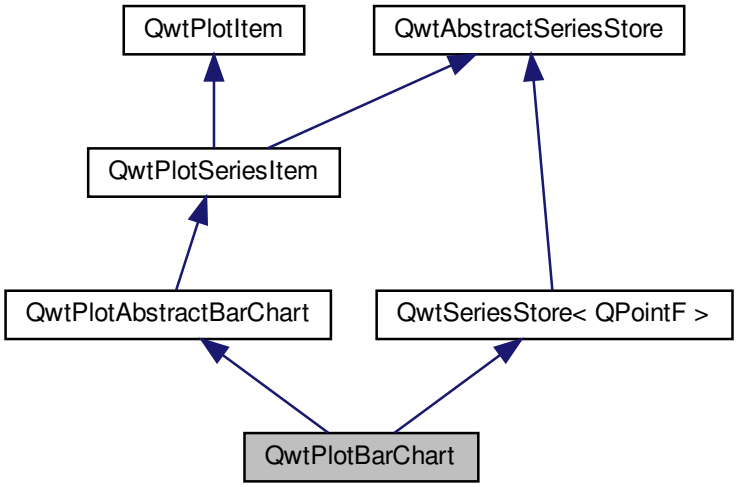
Definition at line 952 of file qwt_plot_abstract_canvas.cpp.

14.74 QwtPlotBarChart Class Reference

QwtPlotBarChart displays a series of a values as bars.

`#include <qwt_plot_barchart.h>`

Inheritance diagram for QwtPlotBarChart:



Public Types

- enum LegendMode { LegendChartTitle , LegendBarTitles }

  Legend modes.

Public Member Functions

- QwtPlotBarChart (const QString &title=QString())
- QwtPlotBarChart (const QwtText &title)
- virtual ~QwtPlotBarChart ()

  Destructor.
- virtual int rtti () const override
- void setSamples (const QVector< QPointF > &)
- void setSamples (const QVector< double > &)
- void setSamples (QwtSeriesData< QPointF > *)
- void setSymbol (QwtColumnSymbol *)

Assign a symbol.

- const QwtColumnSymbol * symbol () const
- void setLegendMode (LegendMode)
- LegendMode legendMode () const
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual QRectF boundingRect () const override
- virtual QwtColumnSymbol * specialSymbol (int sampleIndex, const QPointF &) const
- virtual QwtText barTitle (int sampleIndex) const

  Return the title of a bar.

Protected Member Functions

- virtual void drawSample (QPainter *painter, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, const QwtInterval &boundingInterval, int index, const QPointF &sample) const
- virtual void drawBar (QPainter *, int sampleIndex, const QPointF &sample, const QwtColumnRect &) const
- QwtColumnRect columnRect (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, const QwtInterval &boundingInterval, const QPointF &sample) const
- QList< QwtLegendData > legendData () const override

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

14.74.1 Detailed Description

QwtPlotBarChart displays a series of a values as bars.

Each bar might be customized individually by implementing a specialSymbol(). Otherwise it is rendered using a default symbol.

Depending on its orientation() the bars are displayed horizontally or vertically. The bars cover the interval between the baseline() and the value.

By activating the LegendBarTitles mode each sample will have its own entry on the legend.

The most common use case of a bar chart is to display a list of y coordinates, where the x coordinate is simply the index in the list. But for other situations (f.e. when values are related to dates) it is also possible to set x coordinates explicitly.

See also

QwtPlotMultiBarChart, QwtPlotHistogram, QwtPlotCurve::Sticks, QwtPlotSeriesItem::orientation(), QwtPlotAbstractBarChart::ba

Definition at line 41 of file qwt_plot_barchart.h.

14.74.2 Member Enumeration Documentation

14.74.2.1 LegendMode `enum QwtPlotBarChart::LegendMode`

Legend modes.

The default setting is QwtPlotBarChart::LegendChartTitle.

See also

setLegendMode(), legendMode()

Enumerator

LegendChartTitle	One entry on the legend showing the default symbol and the title() of the chart **See also** QwtPlotItem::title()
LegendBarTitles	One entry for each value showing the individual symbol of the corresponding bar and the bar title. **See also** specialSymbol(), barTitle()

Definition at line 52 of file qwt_plot_barchart.h.

14.74.3 Constructor & Destructor Documentation

14.74.3.1 QwtPlotBarChart() [1/2] QwtPlotBarChart::QwtPlotBarChart (const QString & *title* = *QString()*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 51 of file qwt_plot_barchart.cpp.

14.74.3.2 QwtPlotBarChart() [2/2] QwtPlotBarChart::QwtPlotBarChart (const QwtText & *title*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 41 of file qwt_plot_barchart.cpp.

14.74.4 Member Function Documentation

14.74.4.1 barTitle()

```
QwtText QwtPlotBarChart::barTitle (
        int sampleIndex ) const  [virtual]
```

Return the title of a bar.

In LegendBarTitles mode the title is displayed on the legend entry corresponding to a bar.

The default implementation is a dummy, that is intended to be overloaded.

Parameters

sampleIndex	Index of the bar

Returns

An empty text

See also

LegendBarTitles

Definition at line 399 of file qwt_plot_barchart.cpp.

14.74.4.2 boundingRect()

```
QRectF QwtPlotBarChart::boundingRect ( ) const  [override], [virtual]
```

Returns

Bounding rectangle of all samples. For an empty series the rectangle is invalid.

Reimplemented from QwtPlotSeriesItem.

Definition at line 186 of file qwt_plot_barchart.cpp.

14.74.4.3 columnRect()

```
QwtColumnRect QwtPlotBarChart::columnRect (
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect,
        const QwtInterval & boundingInterval,
        const QPointF & sample ) const  [protected]
```

Calculate the geometry of a bar in widget coordinates

Parameters

xMap	x map
yMap	y map
canvasRect	Contents rect of the canvas
boundingInterval	Bounding interval of sample values
sample	Value of the sample

Returns

Geometry of the column

Definition at line 262 of file qwt_plot_barchart.cpp.

14.74.4.4 drawBar()

```
void QwtPlotBarChart::drawBar (
    QPainter * painter,
    int sampleIndex,
    const QPointF & sample,
    const QwtColumnRect & rect ) const  [protected], [virtual]
```

Draw a bar

Parameters

painter	Painter
sampleIndex	Index of the sample represented by the bar
sample	Value of the sample
rect	Bounding rectangle of the bar

Definition at line 341 of file qwt_plot_barchart.cpp.

14.74.4.5 drawSample()

```
void QwtPlotBarChart::drawSample (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    const QwtInterval & boundingInterval,
    int index,
    const QPointF & sample ) const  [protected], [virtual]
```

Draw a sample

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rect of the canvas
boundingInterval	Bounding interval of sample values
index	Index of the sample
sample	Value of the sample

See also

drawSeries()

Definition at line 322 of file qwt_plot_barchart.cpp.

14.74.4.6 drawSeries()

```
void QwtPlotBarChart::drawSeries (
        QPainter * painter,
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect,
        int from,
        int to ) const  [override], [virtual]
```

Draw an interval of the bar chart

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rect of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

See also

drawSymbols()

Implements QwtPlotSeriesItem.

Definition at line 223 of file qwt_plot_barchart.cpp.

14.74.4.7 legendData()

```
QList< QwtLegendData > QwtPlotBarChart::legendData ( ) const  [override],
[protected], [virtual]
```

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

In case of LegendBarTitles an entry for each bar is returned, otherwise the chart is represented like any other plot item from its title() and the legendIcon().

Returns

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

See also

title(), setLegendMode(), barTitle(), QwtLegend, QwtPlotLegendItem

Reimplemented from QwtPlotItem.

Definition at line 416 of file qwt_plot_barchart.cpp.

14.74.4.8 legendIcon() `QwtGraphic QwtPlotBarChart::legendIcon ( int index, const QSizeF & size ) const [override], [protected], [virtual]`

Returns

Icon representing a bar or the chart on the legend

When the legendMode() is LegendBarTitles the icon shows the bar corresponding to index - otherwise the bar displays the default symbol.

Parameters

index	Index of the legend entry
size	Icon size

See also

setLegendMode(), drawBar(), QwtPlotItem::setLegendIconSize(), QwtPlotItem::legendData()

Reimplemented from QwtPlotItem.

Definition at line 462 of file qwt_plot_barchart.cpp.

14.74.4.9 legendMode() `QwtPlotBarChart::LegendMode QwtPlotBarChart::legendMode ( ) const`

Returns

Legend mode

See also

setLegendMode()

Definition at line 177 of file qwt_plot_barchart.cpp.

14.74.4.10 rtti() `int QwtPlotBarChart::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotBarChart

Reimplemented from QwtPlotItem.

Definition at line 70 of file qwt_plot_barchart.cpp.

14.74.4.11 setLegendMode() `void QwtPlotBarChart::setLegendMode ( LegendMode mode )`

Set the mode that decides what to display on the legend

In case of LegendBarTitles barTitle() needs to be overloaded to return individual titles for each bar.

Parameters

mode	New mode

See also

legendMode(), legendData(), barTitle(), QwtPlotItem::ItemAttribute

Definition at line 164 of file qwt_plot_barchart.cpp.

14.74.4.12 setSamples() [1/3]

```
void QwtPlotBarChart::setSamples (
    const QVector< double > & samples )
```

Initialize data with an array of doubles

The indices in the array are taken as x coordinate, while the doubles are interpreted as y values.

Parameters

samples	Vector of y coordinates

Note

QVector is implicitly shared

Definition at line 97 of file qwt_plot_barchart.cpp.

14.74.4.13 setSamples() [2/3]

```
void QwtPlotBarChart::setSamples (
    const QVector< QPointF > & samples )
```

Initialize data with an array of points

Parameters

samples	Vector of points

Note

QVector is implicitly shared

QPolygonF is derived from QVector<QPointF>

Definition at line 82 of file qwt_plot_barchart.cpp.

14.74.4.14 setSamples() [3/3] void QwtPlotBarChart::setSamples (QwtSeriesData< QPointF > * *data*)

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 119 of file qwt_plot_barchart.cpp.

14.74.4.15 setSymbol() void QwtPlotBarChart::setSymbol (QwtColumnSymbol * *symbol*)

Assign a symbol.

The bar chart will take the ownership of the symbol, hence the previously set symbol will be delete by setting a new one. If `symbol` is `NULL` no symbol will be drawn.

Parameters

symbol	Symbol

See also

symbol()

Definition at line 134 of file qwt_plot_barchart.cpp.

14.74.4.16 specialSymbol() QwtColumnSymbol * QwtPlotBarChart::specialSymbol (int *sampleIndex*, const QPointF & *sample*) const [virtual]

Needs to be overloaded to return a non default symbol for a specific sample

Parameters

sampleIndex	Index of the sample represented by the bar
sample	Value of the sample

Returns

NULL, indicating to use the default symbol

Definition at line 377 of file qwt_plot_barchart.cpp.

14.74.4.17 symbol() `const QwtColumnSymbol * QwtPlotBarChart::symbol ( ) const`

Returns

Current symbol or NULL, when no symbol has been assigned

See also

setSymbol()

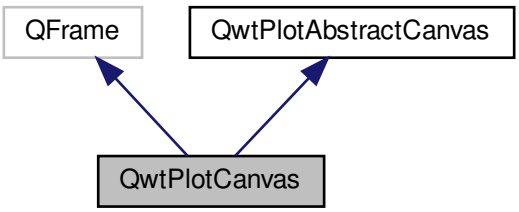
Definition at line 150 of file qwt_plot_barchart.cpp.

14.75 QwtPlotCanvas Class Reference

Canvas of a QwtPlot.

`#include <qwt_plot_canvas.h>`

Inheritance diagram for QwtPlotCanvas:



Public Types

- enum PaintAttribute { BackingStore = 1 , Opaque = 2 , HackStyledBackground = 4 , ImmediatePaint = 8 }
  Paint attributes.
- typedef QFlags< PaintAttribute > PaintAttributes

Public Slots

- void replot ()

Public Member Functions

- QwtPlotCanvas (QwtPlot *=NULL)

  Constructor.
- virtual ~QwtPlotCanvas ()

  Destructor.
- void setPaintAttribute (PaintAttribute, bool on=true)

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

  Invalidate the internal backing store.
- virtual bool event (QEvent *) override
- Q_INVOKABLE QPainterPath borderPath (const QRect &) const

Protected Member Functions

- virtual void paintEvent (QPaintEvent *) override
- virtual void resizeEvent (QResizeEvent *) override
- virtual void drawBorder (QPainter *) override

14.75.1 Detailed Description

Canvas of a QwtPlot.

Canvas is the widget where all plot items are displayed

See also

QwtPlot::setCanvas(), QwtPlotGLCanvas, QwtPlotOpenGLCanvas

Definition at line 29 of file qwt_plot_canvas.h.

14.75.2 Member Typedef Documentation

14.75.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotCanvas::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 102 of file qwt_plot_canvas.h.

14.75.3 Member Enumeration Documentation

14.75.3.1 PaintAttribute `enum QwtPlotCanvas::PaintAttribute`

Paint attributes.

The default setting enables BackingStore and Opaque.

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

BackingStore	Paint double buffered reusing the content of the pixmap buffer when possible. Using a backing store might improve the performance significantly, when working with widget overlays (like rubber bands). Disabling the cache might improve the performance for incremental paints (using QwtPlotDirectPainter). **See also** backingStore(), invalidateBackingStore()
Opaque	Try to fill the complete contents rectangle of the plot canvas. When using styled backgrounds Qt assumes, that the canvas doesn't fill its area completely (f.e because of rounded borders) and fills the area below the canvas. When this is done with gradients it might result in a serious performance bottleneck - depending on the size. When the Opaque attribute is enabled the canvas tries to identify the gaps with some heuristics and to fill those only. **Warning** Will not work for semitransparent backgrounds
HackStyledBackground	Try to improve painting of styled backgrounds. QwtPlotCanvas supports the box model attributes for customizing the layout with style sheets. Unfortunately the design of Qt style sheets has no concept how to handle backgrounds with rounded corners - beside of padding. When HackStyledBackground is enabled the plot canvas tries to separate the background from the background border by reverse engineering to paint the background before and the border after the plot items. In this order the border gets perfectly antialiased and you can avoid some pixel artifacts in the corners.
ImmediatePaint	When ImmediatePaint is set replot() calls repaint() instead of update(). **See also** replot(), QWidget::repaint(), QWidget::update()

Definition at line 44 of file qwt_plot_canvas.h.

14.75.4 Constructor & Destructor Documentation

14.75.4.1 QwtPlotCanvas() `QwtPlotCanvas::QwtPlotCanvas ( QwtPlot * plot = NULL ) [explicit]`

Constructor.

Parameters

plot	Parent plot widget

See also

QwtPlot::setCanvas()

Definition at line 41 of file qwt_plot_canvas.cpp.

14.75.5 Member Function Documentation

14.75.5.1 backingStore() `const QPixmap * QwtPlotCanvas::backingStore ( ) const`

Returns

Backing store, might be null

Definition at line 133 of file qwt_plot_canvas.cpp.

14.75.5.2 borderPath() `QPainterPath QwtPlotCanvas::borderPath ( const QRect & rect ) const`

Calculate the painter path for a styled or rounded border

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

Parameters

rect	Bounding rectangle of the canvas

Returns

Painter path, that can be used for clipping

Definition at line 320 of file qwt_plot_canvas.cpp.

14.75.5.3 drawBorder() `void QwtPlotCanvas::drawBorder ( QPainter * painter ) [override], [protected], [virtual]`

Draw the border of the plot canvas

Parameters

painter	Painter

See also

setBorderRadius()

Reimplemented from QwtPlotAbstractCanvas.

Definition at line 276 of file qwt_plot_canvas.cpp.

14.75.5.4 event() `bool QwtPlotCanvas::event ( QEvent * event ) [override], [virtual]`

Qt event handler for QEvent::PolishRequest and QEvent::StyleChange

Parameters

event	Qt Event

Returns

See QFrame::event()

Definition at line 151 of file qwt_plot_canvas.cpp.

14.75.5.5 paintEvent() `void QwtPlotCanvas::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Paint event

Parameters

event	Paint event

Definition at line 178 of file qwt_plot_canvas.cpp.

14.75.5.6 replot `void QwtPlotCanvas::replot ( ) [slot]`

Invalidate the paint cache and repaint the canvas

See also

invalidatePaintCache()

Definition at line 301 of file qwt_plot_canvas.cpp.

14.75.5.7 resizeEvent()

```
void QwtPlotCanvas::resizeEvent (
    QResizeEvent * event )  [override], [protected], [virtual]
```

Resize event

Parameters

event	Resize event

Definition at line 291 of file qwt_plot_canvas.cpp.

14.75.5.8 setPaintAttribute()

```
void QwtPlotCanvas::setPaintAttribute (
    PaintAttribute attribute,
    bool on = true )
```

Changing the paint attributes.

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute(), backingStore()

Definition at line 70 of file qwt_plot_canvas.cpp.

14.75.5.9 testPaintAttribute()

```
bool QwtPlotCanvas::testPaintAttribute (
    PaintAttribute attribute ) const
```

Test whether a paint attribute is enabled

Parameters

attribute	Paint attribute

Returns

true, when attribute is enabled

See also

setPaintAttribute()

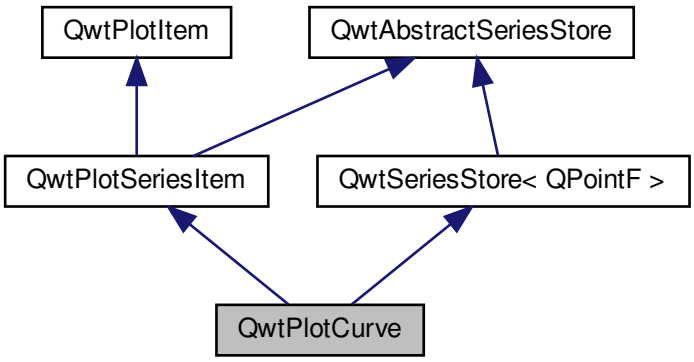
Definition at line 127 of file qwt_plot_canvas.cpp.

14.76 QwtPlotCurve Class Reference

A plot item, that represents a series of points.

`#include <qwt_plot_curve.h>`

Inheritance diagram for QwtPlotCurve:



Public Types

- enum CurveStyle {
  NoCurve = -1 , Lines , Sticks , Steps ,
  Dots , UserCurve = 100 }
- enum CurveAttribute { Inverted = 0x01 , Fitted = 0x02 }
- enum LegendAttribute { LegendNoAttribute = 0x00 , LegendShowLine = 0x01 , LegendShowSymbol = 0x02 , LegendShowBrush = 0x04 }
- enum PaintAttribute {
  ClipPolygons = 0x01 , FilterPoints = 0x02 , MinimizeMemory = 0x04 , ImageBuffer = 0x08 ,
  FilterPointsAggressive = 0x10 }
- typedef QFlags< CurveAttribute > CurveAttributes
- typedef QFlags< LegendAttribute > LegendAttributes
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPlotCurve (const QString &title=QString())
- QwtPlotCurve (const QwtText &title)
- virtual ~QwtPlotCurve ()

  Destructor.
- virtual int rtti () const override
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setLegendAttribute (LegendAttribute, bool on=true)
- bool testLegendAttribute (LegendAttribute) const
- void setLegendAttributes (LegendAttributes)

- LegendAttributes legendAttributes () const
- void setRawSamples (const double *xData, const double *yData, int size)

  Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.
- void setRawSamples (const float *xData, const float *yData, int size)

  Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.
- void setRawSamples (const double *yData, int size)

  Initialize the data by pointing to a memory block which is not managed by QwtPlotCurve.
- void setRawSamples (const float *yData, int size)

  Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.
- void setSamples (const double *xData, const double *yData, int size)
- void setSamples (const float *xData, const float *yData, int size)
- void setSamples (const double *yData, int size)
- void setSamples (const float *yData, int size)
- void setSamples (const QVector< double > &yData)
- void setSamples (const QVector< float > &yData)
- void setSamples (const QVector< double > &xData, const QVector< double > &yData)

  Initialize data with x- and y-arrays (explicitly shared)
- void setSamples (const QVector< float > &xData, const QVector< float > &yData)

  Initialize data with x- and y-arrays (explicitly shared)
- void setSamples (const QVector< QPointF > &)
- void setSamples (QwtSeriesData< QPointF > *)
- virtual int closestPoint (const QPointF &pos, double *dist=NULL) const
- double minXValue () const

  boundingRect().left()
- double maxXValue () const

  boundingRect().right()
- double minYValue () const

  boundingRect().top()
- double maxYValue () const

  boundingRect().bottom()
- void setCurveAttribute (CurveAttribute, bool on=true)
- bool testCurveAttribute (CurveAttribute) const
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)
- const QPen & pen () const
- void setBrush (const QBrush &)

  Assign a brush.
- const QBrush & brush () const
- void setBaseline (double)

  Set the value of the baseline.
- double baseline () const
- void setStyle (CurveStyle style)
- CurveStyle style () const
- void setSymbol (QwtSymbol *)

  Assign a symbol.
- const QwtSymbol * symbol () const
- void setCurveFitter (QwtCurveFitter *)
- QwtCurveFitter * curveFitter () const
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override

Protected Member Functions

- void init ()

  Initialize internal members.
- virtual void drawCurve (QPainter *, int style, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const

  Draw the line part (without symbols) of a curve interval.
- virtual void drawSymbols (QPainter *, const QwtSymbol &, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void drawLines (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const

  Draw lines.
- virtual void drawSticks (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void drawDots (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void drawSteps (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void fillCurve (QPainter *, const QwtScaleMap &, const QwtScaleMap &, const QRectF &canvasRect, QPolygonF &) const
- void closePolyline (QPainter *, const QwtScaleMap &, const QwtScaleMap &, QPolygonF &) const

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

14.76.1 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

a) Assign curve properties When a curve is created, it is configured to draw black solid lines with in QwtPlotCurve::Lines style and no symbols. You can change this by calling setPen(), setStyle() and setSymbol().

b) Connect/Assign data. QwtPlotCurve gets its points using a QwtSeriesData object offering a bridge to the real storage of the points (like QAbstractItemModel). There are several convenience classes derived from QwtSeriesData, that also store the points inside (like QStandardItemModel). QwtPlotCurve also offers a couple of variations of setSamples(), that build QwtSeriesData objects from arrays internally.

c) Attach the curve to a plot See QwtPlotItem::attach()

Example:

see examples/bode

See also

QwtPointSeriesData, QwtSymbol, QwtScaleMap

Definition at line 56 of file qwt_plot_curve.h.

14.76.2 Member Typedef Documentation

14.76.2.1 CurveAttributes `typedef QFlags<CurveAttribute > QwtPlotCurve::CurveAttributes`

An ORed combination of CurveAttribute values.

Definition at line 133 of file qwt_plot_curve.h.

14.76.2.2 LegendAttributes `typedef QFlags<LegendAttribute > QwtPlotCurve::LegendAttributes`

An ORed combination of LegendAttribute values.

Definition at line 168 of file qwt_plot_curve.h.

14.76.2.3 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotCurve::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 234 of file qwt_plot_curve.h.

14.76.3 Member Enumeration Documentation

14.76.3.1 CurveAttribute `enum QwtPlotCurve::CurveAttribute`

Attribute for drawing the curve

See also

setCurveAttribute(), testCurveAttribute(), curveFitter()

Enumerator

Inverted	For QwtPlotCurve::Steps only. Draws a step function from the right to the left.
Fitted	Only in combination with QwtPlotCurve::Lines A QwtCurveFitter tries to interpolate/smooth the curve, before it is painted. **Note** Curve fitting requires temporary memory for calculating coefficients and additional points. If painting in QwtPlotCurve::Fitted mode is slow it might be better to fit the points, before they are passed to QwtPlotCurve.

Definition at line 112 of file qwt_plot_curve.h.

14.76.3.2 CurveStyle enum QwtPlotCurve::CurveStyle

Curve styles.

See also

setStyle(), style()

Enumerator

NoCurve	Don't draw a curve. Note: This doesn't affect the symbols.
Lines	Connect the points with straight lines. The lines might be interpolated depending on the 'Fitted' attribute. Curve fitting can be configured using setCurveFitter().
Sticks	Draw vertical or horizontal sticks (depending on the orientation()) 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 QwtPlotCurve::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 QwtPlotCurve::NoStyle style and a symbol painting a point.
UserCurve	Styles >= QwtPlotCurve::UserCurve are reserved for derived classes of QwtPlotCurve that overload drawCurve() with additional application specific curve types.

Definition at line 65 of file qwt_plot_curve.h.

14.76.3.3 LegendAttribute enum QwtPlotCurve::LegendAttribute

Attributes how to represent the curve on the legend

See also

setLegendAttribute(), testLegendAttribute(), QwtPlotItem::legendData(), legendIcon()

Enumerator

LegendNoAttribute	QwtPlotCurve tries to find a color representing the curve and paints a rectangle with it.
LegendShowLine	If the style() is not QwtPlotCurve::NoCurve a line is painted with the curve pen().
LegendShowSymbol	If the curve has a valid symbol it is painted.
LegendShowBrush	If the curve has a brush a rectangle filled with the curve brush() is painted.

Definition at line 142 of file qwt_plot_curve.h.

14.76.3.4 PaintAttribute `enum QwtPlotCurve::PaintAttribute`

Attributes to modify the drawing algorithm. The default setting enables ClipPolygons | FilterPoints

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

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
FilterPoints	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 algorithms are implemented.
MinimizeMemory	Minimize memory usage that is temporarily needed for the translated points, before they get painted. This might slow down the performance of painting
ImageBuffer	Render the points to a temporary image and paint the image. This is a very special optimization for Dots style, when having a huge amount of points. With a reasonable number of points QPainter::drawPoints() will be faster.
FilterPointsAggressive	More aggressive point filtering trying to filter out intermediate points, accepting minor visual differences. Has only an effect, when drawing the curve to a paint device in integer coordinates (f.e. all widgets on screen) using the fact, that consecutive points are often mapped to the same x or y coordinate. Each chunk of samples mapped to the same coordinate can be reduced to 4 points (first, min, max last). In the worst case the polygon to be rendered will be 4 times the width of the plot canvas. The algorithm is very fast and effective for huge datasets, and can be used inside a replot cycle. **Note** Implemented for QwtPlotCurve::Lines only As this algo replaces many small lines by a long one a nasty bug of the raster paint engine (Qt 4.8, Qt 5.1 - 5.3) becomes more dominant. For these versions the bug can be worked around by enabling the QwtPainter::polylineSplitting() mode.

Definition at line 176 of file qwt_plot_curve.h.

14.76.4 Constructor & Destructor Documentation

14.76.4.1 QwtPlotCurve() [1/2] `QwtPlotCurve::QwtPlotCurve ( const QString & title = QString() ) [explicit]`

Constructor

Parameters

title	Title of the curve

Definition at line 120 of file qwt_plot_curve.cpp.

14.76.4.2 QwtPlotCurve() [2/2] QwtPlotCurve::QwtPlotCurve (const QwtText & *title*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 110 of file qwt_plot_curve.cpp.

14.76.5 Member Function Documentation

14.76.5.1 baseline() double QwtPlotCurve::baseline ( ) const

Returns

Value of the baseline

See also

setBaseline()

Definition at line 1035 of file qwt_plot_curve.cpp.

14.76.5.2 brush() const QBrush & QwtPlotCurve::brush ( ) const

Returns

Brush used to fill the area between lines and the baseline

See also

setBrush(), setBaseline(), baseline()

Definition at line 363 of file qwt_plot_curve.cpp.

14.76.5.3 closePolyline() void QwtPlotCurve::closePolyline (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, QPolygonF & *polygon*) const [protected]

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

Parameters

painter	Painter
xMap	X map
yMap	Y map
polygon	Polygon to be completed

Definition at line 929 of file qwt_plot_curve.cpp.

14.76.5.4 closestPoint()

```
int QwtPlotCurve::closestPoint (
    const QPointF & pos,
    double * dist = NULL ) const  [virtual]
```

Find the closest curve point for a specific position

Parameters

pos	Position, where to look for the closest curve point
dist	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

Definition at line 1051 of file qwt_plot_curve.cpp.

14.76.5.5 curveFitter()

```
QwtCurveFitter * QwtPlotCurve::curveFitter ( ) const
```

Get the curve fitter. If curve fitting is disabled NULL is returned.

Returns

Curve fitter

See also

setCurveFitter(), Fitted

Definition at line 872 of file qwt_plot_curve.cpp.

14.76.5.6 drawCurve()

```
void QwtPlotCurve::drawCurve (
    QPainter * painter,
    int style,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

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

Parameters

painter	Painter
style	curve style, see QwtPlotCurve::CurveStyle
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	index of the first point to be painted
to	index of the last point to be painted

See also

draw(), drawDots(), drawLines(), drawSteps(), drawSticks()

Definition at line 429 of file qwt_plot_curve.cpp.

14.76.5.7 drawDots()

```
void QwtPlotCurve::drawDots (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw dots

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	index of the first point to be painted
to	index of the last point to be painted

See also

draw(), drawCurve(), drawSticks(), drawLines(), drawSteps()

Definition at line 640 of file qwt_plot_curve.cpp.

14.76.5.8 drawLines()

```
void QwtPlotCurve::drawLines (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw lines.

If the CurveAttribute Fitted is enabled a QwtCurveFitter tries to interpolate/smooth the curve, before it is painted.

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	index of the first point to be painted
to	index of the last point to be painted

See also

setCurveAttribute(), setCurveFitter(), draw(), drawLines(), drawDots(), drawSteps(), drawSticks()

Definition at line 476 of file qwt_plot_curve.cpp.

14.76.5.9 drawSeries()

```
void QwtPlotCurve::drawSeries (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [override], [virtual]
```

Draw an interval of the curve

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

See also

drawCurve(), drawSymbols(),

Implements QwtPlotSeriesItem.

Definition at line 381 of file qwt_plot_curve.cpp.

14.76.5.10 drawSteps()

```
void QwtPlotCurve::drawSteps (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw step function

The direction of the steps depends on Inverted attribute.

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	index of the first point to be painted
to	index of the last point to be painted

See also

CurveAttribute, setCurveAttribute(), draw(), drawCurve(), drawDots(), drawLines(), drawSticks()

Definition at line 741 of file qwt_plot_curve.cpp.

14.76.5.11 drawSticks()

```
void QwtPlotCurve::drawSticks (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw sticks

Parameters

painter	Painter
xMap	x map

Parameters

yMap	y map
canvasRect	Contents rectangle of the canvas
from	index of the first point to be painted
to	index of the last point to be painted

See also

draw(), drawCurve(), drawDots(), drawLines(), drawSteps()

Definition at line 585 of file qwt_plot_curve.cpp.

14.76.5.12 drawSymbols()

```
void QwtPlotCurve::drawSymbols (
    QPainter * painter,
    const QwtSymbol & symbol,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw symbols

Parameters

painter	Painter
symbol	Curve symbol
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted

See also

setSymbol(), drawSeries(), drawCurve()

Definition at line 979 of file qwt_plot_curve.cpp.

14.76.5.13 fillCurve()

```
void QwtPlotCurve::fillCurve (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    QPolygonF & polygon ) const  [protected], [virtual]
```

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

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
polygon	Polygon - will be modified !

See also

setBrush(), setBaseline(), setStyle()

Definition at line 889 of file qwt_plot_curve.cpp.

14.76.5.14 legendAttributes()

```
QwtPlotCurve::LegendAttributes QwtPlotCurve::legendAttributes ( ) const
```

Returns

Attributes how to draw the legend icon

See also

setLegendAttributes(), testLegendAttribute()

Definition at line 225 of file qwt_plot_curve.cpp.

14.76.5.15 legendIcon()

```
QwtGraphic QwtPlotCurve::legendIcon (
    int index,
    const QSizeF & size ) const  [override], [virtual]
```

Returns

Icon representing the curve on the legend

Parameters

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

See also

QwtPlotItem::setLegendIconSize(), QwtPlotItem::legendData()

Reimplemented from QwtPlotItem.

Definition at line 1095 of file qwt_plot_curve.cpp.

14.76.5.16 pen() `const QPen & QwtPlotCurve::pen ( ) const`

Returns

Pen used to draw the lines

See also

setPen(), brush()

Definition at line 328 of file qwt_plot_curve.cpp.

14.76.5.17 rtti() `int QwtPlotCurve::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotCurve

Reimplemented from QwtPlotItem.

Definition at line 145 of file qwt_plot_curve.cpp.

14.76.5.18 setBaseline() `void QwtPlotCurve::setBaseline ( double value )`

Set the value of the baseline.

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

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

The default value is 0.0.

Parameters

value	Value of the baseline

See also

baseline(), setBrush(), setStyle(), QwtPlotAbstractSeriesItem::orientation()

Definition at line 1022 of file qwt_plot_curve.cpp.

14.76.5.19 setBrush() `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

brush	New brush

See also

brush(), setBaseline(), baseline()

Definition at line 348 of file qwt_plot_curve.cpp.

14.76.5.20 setCurveAttribute() `void QwtPlotCurve::setCurveAttribute ( CurveAttribute attribute, bool on = true )`

Specify an attribute for drawing the curve

Parameters

attribute	Curve attribute
on	On/Off

/sa testCurveAttribute(), setCurveFitter()

Definition at line 819 of file qwt_plot_curve.cpp.

14.76.5.21 setCurveFitter() `void QwtPlotCurve::setCurveFitter ( QwtCurveFitter * curveFitter )`

Assign a curve fitter

The curve fitter "smooths" the curve points, when the Fitted CurveAttribute is set. setCurveFitter(NULL) also disables curve fitting.

The curve fitter operates on the translated points (= widget coordinates) to be functional for logarithmic scales. Obviously this is less performant for fitting algorithms, that reduce the number of points.

For situations, where curve fitting is used to improve the performance of painting huge series of points it might be better to execute the fitter on the curve points once and to cache the result in the QwtSeriesData object.

Parameters

curveFitter()	Curve fitter

See also

Fitted

Definition at line 858 of file qwt_plot_curve.cpp.

14.76.5.22 setLegendAttribute()

```
void QwtPlotCurve::setLegendAttribute (
    LegendAttribute attribute,
    bool on = true )
```

Specify an attribute how to draw the legend icon

Parameters

attribute	Attribute
on	On/Off /sa testLegendAttribute(). legendIcon()

Definition at line 181 of file qwt_plot_curve.cpp.

14.76.5.23 setLegendAttributes()

```
void QwtPlotCurve::setLegendAttributes (
    LegendAttributes attributes )
```

Specify the attributes how to draw the legend icon

Parameters

attributes	Attributes /sa setLegendAttribute(). legendIcon()

Definition at line 210 of file qwt_plot_curve.cpp.

14.76.5.24 setPaintAttribute()

```
void QwtPlotCurve::setPaintAttribute (
    PaintAttribute attribute,
    bool on = true )
```

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 157 of file qwt_plot_curve.cpp.

14.76.5.25 setPen() [1/2]

```
void QwtPlotCurve::setPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 302 of file qwt_plot_curve.cpp.

14.76.5.26 setPen() [2/2]

```
void QwtPlotCurve::setPen (
    const QPen & pen )
```

Assign a pen

Parameters

pen	New pen

See also

pen(), brush()

Definition at line 313 of file qwt_plot_curve.cpp.

14.76.5.27 setRawSamples() [1/4]

```
void QwtPlotCurve::setRawSamples (
    const double * xData,
    const double * yData,
    int size )
```

Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.

setRawSamples is provided for efficiency. It is important to keep the pointers during the lifetime of the underlying QwtCPointerData class.

Parameters

xData	pointer to x data
yData	pointer to y data
size	size of x and y

See also

QwtCPointerData

Definition at line 1203 of file qwt_plot_curve.cpp.

14.76.5.28 setRawSamples() [2/4]

```
void QwtPlotCurve::setRawSamples (
    const double * yData,
    int size )
```

Initialize the data by pointing to a memory block which is not managed by QwtPlotCurve.

The memory contains the y coordinates, while the index is interpreted as x coordinate.

setRawSamples() is provided for efficiency. It is important to keep the pointers during the lifetime of the underlying QwtCPointerValueData class.

Parameters

yData	pointer to y data
size	size of x and y

See also

QwtCPointerData

Definition at line 1245 of file qwt_plot_curve.cpp.

14.76.5.29 setRawSamples() [3/4]

```
void QwtPlotCurve::setRawSamples (
    const float * xData,
    const float * yData,
    int size )
```

Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.

setRawSamples is provided for efficiency. It is important to keep the pointers during the lifetime of the underlying QwtCPointerData class.

Parameters

xData	pointer to x data
yData	pointer to y data
size	size of x and y

See also

QwtCPointerData

Definition at line 1223 of file qwt_plot_curve.cpp.

14.76.5.30 setRawSamples() [4/4]

```
void QwtPlotCurve::setRawSamples (
    const float * yData,
    int size )
```

Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.

The memory contains the y coordinates, while the index is interpreted as x coordinate.

setRawSamples() is provided for efficiency. It is important to keep the pointers during the lifetime of the underlying QwtCPointerValueData class.

Parameters

yData	pointer to y data
size	size of x and y

See also

QwtCPointerData

Definition at line 1266 of file qwt_plot_curve.cpp.

14.76.5.31 setSamples() [1/10]

```
void QwtPlotCurve::setSamples (
    const double * xData,
```

```
const double * yData,
int size )
```

Set data by copying x- and y-values from specified memory blocks. Contrary to setRawSamples(), this function makes a 'deep copy' of the data.

Parameters

xData	pointer to x values
yData	pointer to y values
size	size of xData and yData

See also

QwtPointArrayData

Definition at line 1282 of file qwt_plot_curve.cpp.

14.76.5.32 setSamples() [2/10]

```
void QwtPlotCurve::setSamples (
    const double * yData,
    int size )
```

Set data by copying y-values from a specified memory block.

The memory contains the y coordinates, while the index is interpreted as x coordinate.

Parameters

yData	y data
size	size of yData

See also

QwtValuePointData

Definition at line 1344 of file qwt_plot_curve.cpp.

14.76.5.33 setSamples() [3/10]

```
void QwtPlotCurve::setSamples (
    const float * xData,
    const float * yData,
    int size )
```

Set data by copying x- and y-values from specified memory blocks. Contrary to setRawSamples(), this function makes a 'deep copy' of the data.

Parameters

xData	pointer to x values
yData	pointer to y values
size	size of xData and yData

See also

QwtPointArrayData

Definition at line 1299 of file qwt_plot_curve.cpp.

14.76.5.34 setSamples() [4/10] void QwtPlotCurve::setSamples (const float * yData, int size)

Set data by copying y-values from a specified memory block.

The vector contains the y coordinates, while the index is interpreted as x coordinate.

Parameters

yData	y data
size	size of yData

See also

QwtValuePointData

Definition at line 1360 of file qwt_plot_curve.cpp.

14.76.5.35 setSamples() [5/10] void QwtPlotCurve::setSamples (const QVector< double > & xData, const QVector< double > & yData)

Initialize data with x- and y-arrays (explicitly shared)

Parameters

xData	x data
yData	y data

See also

QwtPointArrayData

Definition at line 1313 of file qwt_plot_curve.cpp.

14.76.5.36 setSamples() [6/10] void QwtPlotCurve::setSamples (const QVector< double > & *yData*)

Initialize data with an array of y values (explicitly shared)

The vector contains the y coordinates, while the index is the x coordinate.

Parameters

yData	y data

See also

QwtValuePointData

Definition at line 1375 of file qwt_plot_curve.cpp.

14.76.5.37 setSamples() [7/10]

```
void QwtPlotCurve::setSamples (
    const QVector< float > & xData,
    const QVector< float > & yData )
```

Initialize data with x- and y-arrays (explicitly shared)

Parameters

xData	x data
yData	y data

See also

QwtPointArrayData

Definition at line 1327 of file qwt_plot_curve.cpp.

14.76.5.38 setSamples() [8/10]

```
void QwtPlotCurve::setSamples (
    const QVector< float > & yData )
```

Initialize data with an array of y values (explicitly shared)

The vector contains the y coordinates, while the index is the x coordinate.

Parameters

yData	y data

See also

QwtValuePointData

Definition at line 1390 of file qwt_plot_curve.cpp.

14.76.5.39 setSamples() [9/10] void QwtPlotCurve::setSamples (const QVector< QPointF > & *samples*)

Initialize data with an array of points.

Parameters

samples	Vector of points

Note

QVector is implicitly shared

QPolygonF is derived from QVector<QPointF>

Definition at line 1184 of file qwt_plot_curve.cpp.

14.76.5.40 setSamples() [10/10] void QwtPlotCurve::setSamples (QwtSeriesData< QPointF > * *data*)

Assign a series of points

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 1172 of file qwt_plot_curve.cpp.

14.76.5.41 setStyle() void QwtPlotCurve::setStyle (CurveStyle *style*)

Set the curve's drawing style

Parameters

style	Curve style

See also

style()

Definition at line 236 of file qwt_plot_curve.cpp.

14.76.5.42 setSymbol() `void QwtPlotCurve::setSymbol ( QwtSymbol * symbol )`

Assign a symbol.

The curve will take the ownership of the symbol, hence the previously set symbol will be delete by setting a new one. If `symbol` is `NULL` no symbol will be drawn.

Parameters

symbol	Symbol

See also

symbol()

Definition at line 266 of file qwt_plot_curve.cpp.

14.76.5.43 style() `QwtPlotCurve::CurveStyle QwtPlotCurve::style ( ) const`

Returns

Style of the curve

See also

setStyle()

Definition at line 251 of file qwt_plot_curve.cpp.

14.76.5.44 symbol() `const QwtSymbol * QwtPlotCurve::symbol ( ) const`

Returns

Current symbol or NULL, when no symbol has been assigned

See also

setSymbol()

Definition at line 284 of file qwt_plot_curve.cpp.

14.76.5.45 testCurveAttribute() `bool QwtPlotCurve::testCurveAttribute ( CurveAttribute attribute ) const`

Returns

true, if attribute is enabled

See also

setCurveAttribute()

Definition at line 836 of file qwt_plot_curve.cpp.

14.76.5.46 testLegendAttribute() `bool QwtPlotCurve::testLegendAttribute ( LegendAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

setLegendAttribute()

Definition at line 199 of file qwt_plot_curve.cpp.

14.76.5.47 testPaintAttribute() `bool QwtPlotCurve::testPaintAttribute ( PaintAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

setPaintAttribute()

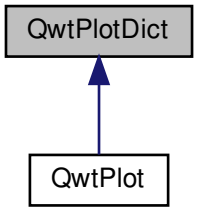
Definition at line 169 of file qwt_plot_curve.cpp.

14.77 QwtPlotDict Class Reference

A dictionary for plot items.

`#include <qwt_plot_dict.h>`

Inheritance diagram for QwtPlotDict:



Public Member Functions

- QwtPlotDict ()
- virtual ~QwtPlotDict ()
- void setAutoDelete (bool)
- bool autoDelete () const
- const QwtPlotItemList & itemList () const

  A QwtPlotItemList of all attached plot items.
- QwtPlotItemList itemList (int rtti) const
- void detachItems (int rtti=QwtPlotItem::Rtti_PlotItem, bool autoDelete=true)

Protected Member Functions

- void insertItem (QwtPlotItem *)
- void removeItem (QwtPlotItem *)

14.77.1 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. QwtPlotDict can be used to get access to all QwtPlotItem items - or all items of a specific type - that are currently on the plot.

See also

QwtPlotItem::attach(), QwtPlotItem::detach(), QwtPlotItem::z()

Definition at line 32 of file qwt_plot_dict.h.

14.77.2 Constructor & Destructor Documentation

14.77.2.1 QwtPlotDict() `QwtPlotDict::QwtPlotDict ( ) [explicit]`

Constructor

Auto deletion is enabled.

See also

setAutoDelete(), QwtPlotItem::attach()

Definition at line 69 of file qwt_plot_dict.cpp.

14.77.2.2 ~QwtPlotDict() `QwtPlotDict::~QwtPlotDict ( ) [virtual]`

Destructor

If autoDelete() is on, all attached items will be deleted

See also

setAutoDelete(), autoDelete(), QwtPlotItem::attach()

Definition at line 81 of file qwt_plot_dict.cpp.

14.77.3 Member Function Documentation

14.77.3.1 autoDelete() `bool QwtPlotDict::autoDelete ( ) const`

Returns

true if auto deletion is enabled

See also

setAutoDelete(), insertItem()

Definition at line 104 of file qwt_plot_dict.cpp.

14.77.3.2 detachItems()

```
void QwtPlotDict::detachItems (
        int rtti = QwtPlotItem::Rtti_PlotItem,
        bool autoDelete = true )
```

Detach items from the dictionary

Parameters

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

Definition at line 138 of file qwt_plot_dict.cpp.

14.77.3.3 insertItem() `void QwtPlotDict::insertItem ( QwtPlotItem * item ) [protected]`

Insert a plot item

Parameters

item	PlotItem

See also

removeItem()

Definition at line 115 of file qwt_plot_dict.cpp.

14.77.3.4 itemList() [1/2] `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.

Definition at line 166 of file qwt_plot_dict.cpp.

14.77.3.5 itemList() [2/2] `QwtPlotItemList QwtPlotDict::itemList ( int rtti ) const`

Returns

List of all attached plot items of a specific type.

Parameters

rtti	See QwtPlotItem::RttiValues

See also

QwtPlotItem::rtti()

Definition at line 176 of file qwt_plot_dict.cpp.

14.77.3.6 removeItem() `void QwtPlotDict::removeItem ( QwtPlotItem * item ) [protected]`

Remove a plot item

Parameters

item	PlotItem

See also

insertItem()

Definition at line 126 of file qwt_plot_dict.cpp.

14.77.3.7 setAutoDelete() `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(), insertItem()

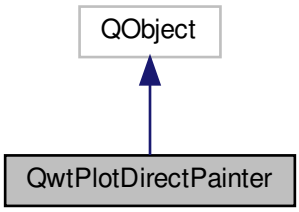
Definition at line 95 of file qwt_plot_dict.cpp.

14.78 QwtPlotDirectPainter Class Reference

Painter object trying to paint incrementally.

`#include <qwt_plot_directpainter.h>`

Inheritance diagram for QwtPlotDirectPainter:



Public Types

- enum Attribute { AtomicPainter = 0x01 , FullRepaint = 0x02 , CopyBackingStore = 0x04 }
  Paint attributes.
- typedef QFlags< Attribute > Attributes

Public Member Functions

- QwtPlotDirectPainter (QObject *parent=NULL)
  Constructor.
- virtual ~QwtPlotDirectPainter ()
  Destructor.
- void setAttribute (Attribute, bool on)
- bool testAttribute (Attribute) const
- void setClipping (bool)
- bool hasClipping () const
- void setClipRegion (const QRegion &)
  Assign a clip region and enable clipping.
- QRegion clipRegion () const
- void drawSeries (QwtPlotSeriesItem *, int from, int to)
  Draw a set of points of a seriesItem.
- void reset ()
  Close the internal QPainter.
- virtual bool eventFilter (QObject *, QEvent *) override
  Event filter.

14.78.1 Detailed Description

Painter object trying to paint incrementally.

Often applications want to display samples while they are collected. When there are too many samples complete replots will be expensive to be processed in a collection cycle.

QwtPlotDirectPainter offers an API to paint subsets (f.e all additions points) without erasing/repainting the plot canvas.

On certain environments it might be important to calculate a proper clip region before painting. F.e. for Qt Embedded only the clipped part of the backing store will be copied to a (maybe unaccelerated) frame buffer.

Warning

Incremental painting will only help when no replot is triggered by another operation (like changing scales) and nothing needs to be erased.

Definition at line 39 of file qwt_plot_directpainter.h.

14.78.2 Member Typedef Documentation

14.78.2.1 Attributes `typedef QFlags<Attribute > QwtPlotDirectPainter::Attributes`

An ORed combination of Attribute values.

Definition at line 73 of file qwt_plot_directpainter.h.

14.78.3 Member Enumeration Documentation

14.78.3.1 Attribute `enum QwtPlotDirectPainter::Attribute`

Paint attributes.

See also

setAttribute(), testAttribute(), drawSeries()

Enumerator

AtomicPainter	Initializing a QPainter is an expensive operation. When AtomicPainter is set each call of drawSeries() opens/closes a temporary QPainter. Otherwise QwtPlotDirectPainter tries to use the same QPainter as long as possible.
FullRepaint	When FullRepaint is set the plot canvas is explicitly repainted after the samples have been rendered.
CopyBackingStore	When QwtPlotCanvas::BackingStore is enabled the painter has to paint to the backing store and the widget. In certain situations/environments it might be faster to paint to the backing store only and then copy the backing store to the canvas. This flag can also be useful for settings, where Qt fills the the clip region with the widget background.

Definition at line 46 of file qwt_plot_directpainter.h.

14.78.4 Member Function Documentation

14.78.4.1 clipRegion() `QRegion QwtPlotDirectPainter::clipRegion ( ) const`

Returns

Currently set clip region.

See also

setClipRegion(), setClipping(), hasClipping()

Definition at line 151 of file qwt_plot_directpainter.cpp.

14.78.4.2 drawSeries() `void QwtPlotDirectPainter::drawSeries (`

```
QwtPlotSeriesItem * seriesItem,
int from,
int to )
```

Draw a set of points of a seriesItem.

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

Setting plot()->canvas()->setAttribute(Qt::WA_PaintOutsidePaintEvent, true); will result in faster painting, if the paint engine of the canvas widget supports this feature.

Parameters

seriesItem	Item to be painted
from	Index of the first point to be painted
to	Index of the last point to be painted. If to < 0 the series will be painted to its last point.

Definition at line 172 of file qwt_plot_directpainter.cpp.

14.78.4.3 hasClipping() `bool QwtPlotDirectPainter::hasClipping ( ) const`

Returns

true, when clipping is enabled

See also

setClipping(), clipRegion(), setClipRegion()

Definition at line 125 of file qwt_plot_directpainter.cpp.

14.78.4.4 setAttribute()

```
void QwtPlotDirectPainter::setAttribute (
    Attribute attribute,
    bool on )
```

Change an attribute

Parameters

attribute	Attribute to change
on	On/Off

See also

Attribute, testAttribute()

Definition at line 86 of file qwt_plot_directpainter.cpp.

14.78.4.5 setClipping()

```
void QwtPlotDirectPainter::setClipping (
    bool enable )
```

En/Disables clipping

Parameters

enable	Enables clipping is true, disable it otherwise

See also

hasClipping(), clipRegion(), setClipRegion()

Definition at line 116 of file qwt_plot_directpainter.cpp.

14.78.4.6 setClipRegion()

```
void QwtPlotDirectPainter::setClipRegion (
    const QRegion & region )
```

Assign a clip region and enable clipping.

Depending on the environment setting a proper clip region might improve the performance heavily. F.e. on Qt embedded only the clipped part of the backing store will be copied to a (maybe unaccelerated) frame buffer device.

Parameters

region	Clip region

See also

clipRegion(), hasClipping(), setClipping()

Definition at line 141 of file qwt_plot_directpainter.cpp.

14.78.4.7 testAttribute() `bool QwtPlotDirectPainter::testAttribute ( Attribute attribute ) const`

Returns

True, when attribute is enabled

Parameters

attribute	Attribute to be tested

See also

Attribute, setAttribute()

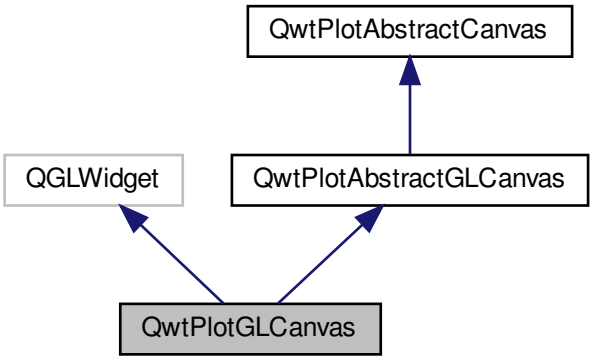
Definition at line 105 of file qwt_plot_directpainter.cpp.

14.79 QwtPlotGLCanvas Class Reference

An alternative canvas for a QwtPlot derived from QGLWidget.

`#include <qwt_plot_glcanvas.h>`

Inheritance diagram for QwtPlotGLCanvas:



Public Slots

- void replot ()

Public Member Functions

- QwtPlotGLCanvas (QwtPlot *=NULL)

  Constructor.
- QwtPlotGLCanvas (const QGLFormat &, QwtPlot *=NULL)

  Constructor.
- virtual ~QwtPlotGLCanvas ()

  Destructor.
- virtual Q_INVOKABLE void invalidateBackingStore () override

  Invalidate the internal backing store.
- Q_INVOKABLE QPainterPath borderPath (const QRect &) const
- virtual bool event (QEvent *) override

Protected Member Functions

- virtual void paintEvent (QPaintEvent *) override
- virtual void initializeGL () override

  No operation - reserved for some potential use in the future.
- virtual void paintGL () override

  Paint the plot.
- virtual void resizeGL (int width, int height) override

  No operation - reserved for some potential use in the future.

Additional Inherited Members

14.79.1 Detailed Description

An alternative canvas for a QwtPlot derived from QGLWidget.

QwtPlotGLCanvas implements the very basics to act as canvas inside of a QwtPlot widget. It might be extended to a full featured alternative to QwtPlotCanvas in a future version of Qwt.

Even if QwtPlotGLCanvas is not derived from QFrame it imitates its API. When using style sheets it supports the box model - beside backgrounds with rounded borders.

Since Qt 5.4 QOpenGLWidget is available, that is used by QwtPlotOpenGLCanvas.

See also

QwtPlot::setCanvas(), QwtPlotCanvas, QwtPlotOpenGLCanvas

Note

With Qt4 you might want to use the QPaintEngine::OpenGL paint engine (see QGL::setPreferredPaintEngine()). On a Linux test system QPaintEngine::OpenGL2 shows very basic problems like translated geometries.

Another way for getting hardware accelerated graphics is using an OpenGL offscreen buffer (QwtPlot↩ Canvas::OpenGLBuffer) with QwtPlotCanvas. Performance is worse, than rendering straight to a QGLWidget, but is usually better integrated into a desktop application.

Definition at line 45 of file qwt_plot_glcanvas.h.

14.79.2 Constructor & Destructor Documentation

14.79.2.1 QwtPlotGLCanvas() [1/2] QwtPlotGLCanvas::QwtPlotGLCanvas (QwtPlot * *plot* = *NULL*) [explicit]

Constructor.

Parameters

plot	Parent plot widget

See also

QwtPlot::setCanvas()

Definition at line 56 of file qwt_plot_glcanvas.cpp.

14.79.2.2 QwtPlotGLCanvas() [2/2]

```
QwtPlotGLCanvas::QwtPlotGLCanvas (
    const QGLFormat & format,
    QwtPlot * plot = NULL )  [explicit]
```

Constructor.

Parameters

format	OpenGL rendering options
plot	Parent plot widget

See also

QwtPlot::setCanvas()

Definition at line 69 of file qwt_plot_glcanvas.cpp.

14.79.3 Member Function Documentation

14.79.3.1 borderPath()

```
QPainterPath QwtPlotGLCanvas::borderPath (
    const QRect & rect ) const
```

Calculate the painter path for a styled or rounded border

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

Parameters

rect	Bounding rectangle of the canvas

Returns

Painter path, that can be used for clipping

Definition at line 157 of file qwt_plot_glcanvas.cpp.

14.79.3.2 event()

```
bool QwtPlotGLCanvas::event (
    QEvent * event )  [override], [virtual]
```

Qt event handler for QEvent::PolishRequest and QEvent::StyleChange

Parameters

event	Qt Event

Returns

See QGLWidget::event()

Definition at line 110 of file qwt_plot_glcanvas.cpp.

14.79.3.3 paintEvent() `void QwtPlotGLCanvas::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Paint event

Parameters

event	Paint event

See also

QwtPlot::drawCanvas()

Definition at line 100 of file qwt_plot_glcanvas.cpp.

14.79.3.4 replot `void QwtPlotGLCanvas::replot ( ) [slot]`

Invalidate the paint cache and repaint the canvas

See also

invalidatePaintCache()

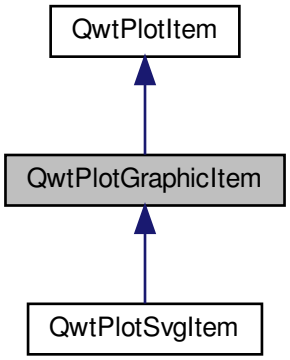
Definition at line 131 of file qwt_plot_glcanvas.cpp.

14.80 QwtPlotGraphicItem Class Reference

A plot item, which displays a recorded sequence of QPainter commands.

`#include <qwt_plot_graphicitem.h>`

Inheritance diagram for QwtPlotGraphicItem:



Public Member Functions

- QwtPlotGraphicItem (const QString &title=QString())

  Constructor.
- QwtPlotGraphicItem (const QwtText &title)

  Constructor.
- virtual ~QwtPlotGraphicItem ()

  Destructor.
- void setGraphic (const QRectF &rect, const QwtGraphic &)
- QwtGraphic graphic () const
- virtual QRectF boundingRect () const override

  Bounding rectangle of the item.
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override
- virtual int rtti () const override

Additional Inherited Members

14.80.1 Detailed Description

A plot item, which displays a recorded sequence of QPainter commands.

QwtPlotGraphicItem renders a sequence of recorded painter commands into a specific plot area. Recording of painter commands can be done manually by QPainter or e.g. QSvgRenderer.

See also

QwtPlotShapeItem, QwtPlotSvgItem

Definition at line 29 of file qwt_plot_graphicitem.h.

14.80.2 Constructor & Destructor Documentation

14.80.2.1 QwtPlotGraphicItem() [1/2] QwtPlotGraphicItem::QwtPlotGraphicItem (const QString & *title* = *QString()*) [explicit]

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

title	Title

Definition at line 32 of file qwt_plot_graphicitem.cpp.

14.80.2.2 QwtPlotGraphicItem() [2/2] QwtPlotGraphicItem::QwtPlotGraphicItem (const QwtText & *title*) [explicit]

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

title	Title

Definition at line 47 of file qwt_plot_graphicitem.cpp.

14.80.3 Member Function Documentation

14.80.3.1 draw() void QwtPlotGraphicItem::draw (QPainter * *painter*, const QwtScaleMap & *xMap*,

```
const QwtScaleMap & yMap,
const QRectF & canvasRect ) const  [override], [virtual]
```

Draw the item

Parameters

painter	Painter
xMap	X-Scale Map
yMap	Y-Scale Map
canvasRect	Contents rect of the plot canvas

Implements QwtPlotItem.

Definition at line 115 of file qwt_plot_graphicitem.cpp.

14.80.3.2 graphic() `QwtGraphic QwtPlotGraphicItem::graphic ( ) const`

Returns

Recorded sequence of painter commands

See also

setGraphic()

Definition at line 96 of file qwt_plot_graphicitem.cpp.

14.80.3.3 rtti() `int QwtPlotGraphicItem::rtti ( ) const  [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotGraphic

Reimplemented from QwtPlotItem.

Definition at line 71 of file qwt_plot_graphicitem.cpp.

14.80.3.4 setGraphic() `void QwtPlotGraphicItem::setGraphic (`

```
const QRectF & rect,
const QwtGraphic & graphic )
```

Set the graphic to be displayed

Parameters

rect	Rectangle in plot coordinates
graphic	Recorded sequence of painter commands

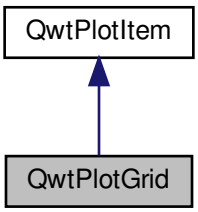
Definition at line 82 of file qwt_plot_graphicitem.cpp.

14.81 QwtPlotGrid Class Reference

A class which draws a coordinate grid.

`#include <qwt_plot_grid.h>`

Inheritance diagram for QwtPlotGrid:



Public Member Functions

- QwtPlotGrid ()

  Enables major grid, disables minor grid.
- virtual ~QwtPlotGrid ()

  Destructor.
- virtual int rtti () const override
- void enableX (bool)

  Enable or disable vertical grid lines.
- bool xEnabled () const
- void enableY (bool)

  Enable or disable horizontal grid lines.
- bool yEnabled () const
- void enableXMin (bool)

  Enable or disable minor vertical grid lines.
- bool xMinEnabled () const
- void enableYMin (bool)

  Enable or disable minor horizontal grid lines.
- bool yMinEnabled () const
- void setXDiv (const QwtScaleDiv &)
- const QwtScaleDiv & xScaleDiv () const

- void setYDiv (const QwtScaleDiv &)
- const QwtScaleDiv & yScaleDiv () const
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)
- void setMajorPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setMajorPen (const QPen &)
- const QPen & majorPen () const
- void setMinorPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setMinorPen (const QPen &)
- const QPen & minorPen () const
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override

  Draw the grid.
- virtual void updateScaleDiv (const QwtScaleDiv &xScaleDiv, const QwtScaleDiv &yScaleDiv) override

Additional Inherited Members

14.81.1 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 grid lines. The locations of the grid lines are determined by the X and Y scale divisions which can be assigned with setXDiv() and setYDiv(). The draw() member draws the grid within a bounding rectangle.

Definition at line 33 of file qwt_plot_grid.h.

14.81.2 Member Function Documentation

14.81.2.1 draw()

```
void QwtPlotGrid::draw (
        QPainter * painter,
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect ) const  [override], [virtual]
```

Draw the grid.

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

Parameters

painter	Painter
xMap	X axis map
yMap	Y axis
canvasRect	Contents rectangle of the plot canvas

Implements QwtPlotItem.

Definition at line 282 of file qwt_plot_grid.cpp.

14.81.2.2 enableX() `void QwtPlotGrid::enableX ( bool on )`

Enable or disable vertical grid lines.

Parameters

on	Enable (true) or disable

See also

Minor grid lines can be enabled or disabled with enableXMin()

Definition at line 76 of file qwt_plot_grid.cpp.

14.81.2.3 enableXMin() `void QwtPlotGrid::enableXMin ( bool on )`

Enable or disable minor vertical grid lines.

Parameters

on	Enable (true) or disable

See also

enableX()

Definition at line 108 of file qwt_plot_grid.cpp.

14.81.2.4 enableY() `void QwtPlotGrid::enableY ( bool on )`

Enable or disable horizontal grid lines.

Parameters

on	Enable (true) or disable

See also

Minor grid lines can be enabled or disabled with enableYMin()

Definition at line 92 of file qwt_plot_grid.cpp.

14.81.2.5 enableYMin() `void QwtPlotGrid::enableYMin ( bool on )`

Enable or disable minor horizontal grid lines.

Parameters

on	Enable (true) or disable

See also

enableY()

Definition at line 124 of file qwt_plot_grid.cpp.

14.81.2.6 majorPen() `const QPen & QwtPlotGrid::majorPen ( ) const`

Returns

the pen for the major grid lines

See also

setMajorPen(), setMinorPen(), setPen()

Definition at line 367 of file qwt_plot_grid.cpp.

14.81.2.7 minorPen() `const QPen & QwtPlotGrid::minorPen ( ) const`

Returns

the pen for the minor grid lines

See also

setMinorPen(), setMajorPen(), setPen()

Definition at line 376 of file qwt_plot_grid.cpp.

14.81.2.8 rtti() `int QwtPlotGrid::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotGrid

Reimplemented from QwtPlotItem.

Definition at line 64 of file qwt_plot_grid.cpp.

14.81.2.9 setMajorPen() [1/2] `void QwtPlotGrid::setMajorPen (`

```
const QColor & color,
qreal width = 0.0,
Qt::PenStyle style = Qt::SolidLine )
```

Build and assign a pen for both major grid lines

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 212 of file qwt_plot_grid.cpp.

14.81.2.10 setMajorPen() [2/2] `void QwtPlotGrid::setMajorPen (`

```
const QPen & pen )
```

Assign a pen for the major grid lines

Parameters

pen	Pen

See also

majorPen(), setMinorPen(), setPen()

Definition at line 223 of file qwt_plot_grid.cpp.

14.81.2.11 setMinorPen() [1/2]

```
void QwtPlotGrid::setMinorPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

Build and assign a pen for the minor grid lines

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 247 of file qwt_plot_grid.cpp.

14.81.2.12 setMinorPen() [2/2]

```
void QwtPlotGrid::setMinorPen (
    const QPen & pen )
```

Assign a pen for the minor grid lines

Parameters

pen	Pen

See also

minorPen(), setMajorPen(), setPen()

Definition at line 258 of file qwt_plot_grid.cpp.

14.81.2.13 setPen() [1/2]

```
void QwtPlotGrid::setPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

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

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 176 of file qwt_plot_grid.cpp.

14.81.2.14 setPen() [2/2] void QwtPlotGrid::setPen (const QPen & *pen*)

Assign a pen for both major and minor grid lines

Parameters

pen	Pen

See also

setMajorPen(), setMinorPen()

Definition at line 187 of file qwt_plot_grid.cpp.

14.81.2.15 setXDiv() void QwtPlotGrid::setXDiv (const QwtScaleDiv & *scaleDiv*)

Assign an x axis scale division

Parameters

scaleDiv	Scale division

Definition at line 140 of file qwt_plot_grid.cpp.

14.81.2.16 setYDiv() void QwtPlotGrid::setYDiv (const QwtScaleDiv & *scaleDiv*)

Assign a y axis division

Parameters

scaleDiv	Scale division

Definition at line 154 of file qwt_plot_grid.cpp.

14.81.2.17 updateScaleDiv()

```
void QwtPlotGrid::updateScaleDiv (
    const QwtScaleDiv & xScaleDiv,
    const QwtScaleDiv & yScaleDiv ) [override], [virtual]
```

Update the grid to changes of the axes scale division

Parameters

xScaleDiv	Scale division of the x-axis
yScaleDiv	Scale division of the y-axis

See also

QwtPlot::updateAxes()

Reimplemented from QwtPlotItem.

Definition at line 438 of file qwt_plot_grid.cpp.

14.81.2.18 xEnabled()

```
bool QwtPlotGrid::xEnabled ( ) const
```

Returns

true if vertical grid lines are enabled

See also

enableX()

Definition at line 385 of file qwt_plot_grid.cpp.

14.81.2.19 xMinEnabled()

```
bool QwtPlotGrid::xMinEnabled ( ) const
```

Returns

true if minor vertical grid lines are enabled

See also

enableXMin()

Definition at line 394 of file qwt_plot_grid.cpp.

14.81.2.20 xScaleDiv() `const QwtScaleDiv & QwtPlotGrid::xScaleDiv ( ) const`

Returns

the scale division of the x axis

Definition at line 419 of file qwt_plot_grid.cpp.

14.81.2.21 yEnabled() `bool QwtPlotGrid::yEnabled ( ) const`

Returns

true if horizontal grid lines are enabled

See also

enableY()

Definition at line 403 of file qwt_plot_grid.cpp.

14.81.2.22 yMinEnabled() `bool QwtPlotGrid::yMinEnabled ( ) const`

Returns

true if minor horizontal grid lines are enabled

See also

enableYMin()

Definition at line 412 of file qwt_plot_grid.cpp.

14.81.2.23 yScaleDiv() `const QwtScaleDiv & QwtPlotGrid::yScaleDiv ( ) const`

Returns

the scale division of the y axis

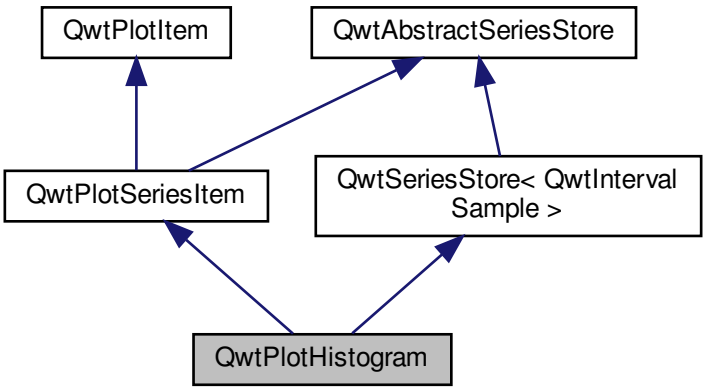
Definition at line 425 of file qwt_plot_grid.cpp.

14.82 QwtPlotHistogram Class Reference

QwtPlotHistogram represents a series of samples, where an interval is associated with a value ($y = f([x1, x2])$).

`#include <qwt_plot_histogram.h>`

Inheritance diagram for QwtPlotHistogram:



Public Types

- enum HistogramStyle { Outline , Columns , Lines , UserStyle = 100 }

Public Member Functions

- QwtPlotHistogram (const QString &title=QString())
- QwtPlotHistogram (const QwtText &title)
- virtual ~QwtPlotHistogram ()

  Destructor.
- virtual int rtti () const override
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)
- const QPen & pen () const
- void setBrush (const QBrush &)
- const QBrush & brush () const
- void setSamples (const QVector< QwtIntervalSample > &)
- void setSamples (QwtSeriesData< QwtIntervalSample > *)
- void setBaseline (double)

  Set the value of the baseline.
- double baseline () const
- void setStyle (HistogramStyle style)
- HistogramStyle style () const
- void setSymbol (const QwtColumnSymbol *)

  Assign a symbol.
- const QwtColumnSymbol * symbol () const
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual QRectF boundingRect () const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override

Protected Member Functions

- virtual QwtColumnRect columnRect (const QwtIntervalSample &, const QwtScaleMap &, const QwtScaleMap &) const
- virtual void drawColumn (QPainter *, const QwtColumnRect &, const QwtIntervalSample &) const
- void drawColumns (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void drawOutline (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void drawLines (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const

14.82.1 Detailed Description

QwtPlotHistogram represents a series of samples, where an interval is associated with a value ($y = f([x1, x2])$).

The representation depends on the style() and an optional symbol() that is displayed for each interval.

Note

The term "histogram" is used in a different way in the areas of digital image processing and statistics. Wikipedia introduces the terms "image histogram" and "color histogram" to avoid confusions. While "image histograms" can be displayed by a QwtPlotCurve there is no applicable plot item for a "color histogram" yet.

See also

QwtPlotBarChart, QwtPlotMultiBarChart

Definition at line 41 of file qwt_plot_histogram.h.

14.82.2 Member Enumeration Documentation

14.82.2.1 HistogramStyle `enum QwtPlotHistogram::HistogramStyle`

Histogram styles. The default style is QwtPlotHistogram::Columns.

See also

setStyle(), style(), setSymbol(), symbol(), setBaseline()

Enumerator

Outline	Draw an outline around the area, that is build by all intervals using the pen() and fill it with the brush(). The outline style requires, that the intervals are in increasing order and not overlapping.
Columns	Draw a column for each interval. When a symbol() has been set the symbol is used otherwise the column is displayed as plain rectangle using pen() and brush().
Lines	Draw a simple line using the pen() for each interval.
UserStyle	Styles >= UserStyle are reserved for derived classes that overload drawSeries() with additional application specific ways to display a histogram.

Definition at line 52 of file qwt_plot_histogram.h.

14.82.3 Constructor & Destructor Documentation

14.82.3.1 QwtPlotHistogram() [1/2] QwtPlotHistogram::QwtPlotHistogram (const QString & *title* = *QString()*) [explicit]

Constructor

Parameters

title	Title of the histogram.

Definition at line 74 of file qwt_plot_histogram.cpp.

14.82.3.2 QwtPlotHistogram() [2/2] QwtPlotHistogram::QwtPlotHistogram (const QwtText & *title*) [explicit]

Constructor

Parameters

title	Title of the histogram.

Definition at line 64 of file qwt_plot_histogram.cpp.

14.82.4 Member Function Documentation

14.82.4.1 baseline() double QwtPlotHistogram::baseline ( ) const

Returns

Value of the baseline

See also

setBaseline()

Definition at line 253 of file qwt_plot_histogram.cpp.

14.82.4.2 boundingRect() QRectF QwtPlotHistogram::boundingRect ( ) const [override], [virtual]

Returns

Bounding rectangle of all samples. For an empty series the rectangle is invalid.

Reimplemented from QwtPlotSeriesItem.

Definition at line 262 of file qwt_plot_histogram.cpp.

14.82.4.3 brush() const QBrush & QwtPlotHistogram::brush ( ) const

Returns

Brush used in a style() depending way.

See also

setPen(), brush()

Definition at line 189 of file qwt_plot_histogram.cpp.

14.82.4.4 columnRect() QwtColumnRect QwtPlotHistogram::columnRect (const QwtIntervalSample & *sample*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const [protected], [virtual]

Calculate the area that is covered by a sample

Parameters

sample	Sample
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.

Returns

Rectangle, that is covered by a sample

Definition at line 602 of file qwt_plot_histogram.cpp.

14.82.4.5 drawColumn()

```
void QwtPlotHistogram::drawColumn (
    QPainter * painter,
    const QwtColumnRect & rect,
    const QwtIntervalSample & sample ) const  [protected], [virtual]
```

Draw a column for a sample in Columns style().

When a symbol() has been set the symbol is used otherwise the column is displayed as plain rectangle using pen() and brush().

Parameters

painter	Painter
rect	Rectangle where to paint the column in paint device coordinates
sample	Sample to be displayed

Note

In applications, where different intervals need to be displayed in a different way (f.e different colors or even using different symbols) it is recommended to overload drawColumn().

Definition at line 653 of file qwt_plot_histogram.cpp.

14.82.4.6 drawColumns()

```
void QwtPlotHistogram::drawColumns (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    int from,
    int to ) const  [protected]
```

Draw a histogram in Columns style()

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the histogram will be painted to its last point.

See also

setStyle(), style(), setSymbol(), drawColumn()

Definition at line 461 of file qwt_plot_histogram.cpp.

14.82.4.7 drawLines()

```
void QwtPlotHistogram::drawLines (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    int from,
    int to ) const  [protected]
```

Draw a histogram in Lines style()

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the histogram will be painted to its last point.

See also

setStyle(), style(), setPen()

Definition at line 493 of file qwt_plot_histogram.cpp.

14.82.4.8 drawOutline()

```
void QwtPlotHistogram::drawOutline (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    int from,
    int to ) const  [protected]
```

Draw a histogram in Outline style()

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the histogram will be painted to its last point.

See also

setStyle(), style()

Warning

The outline style requires, that the intervals are in increasing order and not overlapping.

Definition at line 376 of file qwt_plot_histogram.cpp.

14.82.4.9 drawSeries()

```
void QwtPlotHistogram::drawSeries (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [override], [virtual]
```

Draw a subset of the histogram samples

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the series will be painted to its last sample.

See also

drawOutline(), drawLines(), drawColumns

Implements QwtPlotSeriesItem.

Definition at line 334 of file qwt_plot_histogram.cpp.

14.82.4.10 legendIcon()

```
QwtGraphic QwtPlotHistogram::legendIcon (
    int index,
    const QSizeF & size ) const  [override], [virtual]
```

A plain rectangle without pen using the brush()

Parameters

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

Returns

A graphic displaying the icon

See also

QwtPlotItem::setLegendIconSize(), QwtPlotItem::legendData()

Reimplemented from QwtPlotItem.

Definition at line 688 of file qwt_plot_histogram.cpp.

14.82.4.11 pen() `const QPen & QwtPlotHistogram::pen ( ) const`

Returns

Pen used in a style() depending way.

See also

setPen(), brush()

Definition at line 163 of file qwt_plot_histogram.cpp.

14.82.4.12 rtti() `int QwtPlotHistogram::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotHistogram

Reimplemented from QwtPlotItem.

Definition at line 290 of file qwt_plot_histogram.cpp.

14.82.4.13 setBaseline() `void QwtPlotHistogram::setBaseline ( double value )`

Set the value of the baseline.

Each column representing an QwtIntervalSample is defined by its interval and the interval between baseline and the value of the sample.

The default value of the baseline is 0.0.

Parameters

value	Value of the baseline

See also

baseline()

Definition at line 240 of file qwt_plot_histogram.cpp.

14.82.4.14 setBrush() `void QwtPlotHistogram::setBrush ( const QBrush & brush )`

Assign a brush, that is used in a style() depending way.

Parameters

brush	New brush

See also

pen(), brush()

Definition at line 174 of file qwt_plot_histogram.cpp.

14.82.4.15 setPen() [1/2]

```
void QwtPlotHistogram::setPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 137 of file qwt_plot_histogram.cpp.

14.82.4.16 setPen() [2/2]

```
void QwtPlotHistogram::setPen (
    const QPen & pen )
```

Assign a pen, that is used in a style() depending way.

Parameters

pen	New pen

See also

pen(), brush()

Definition at line 148 of file qwt_plot_histogram.cpp.

14.82.4.17 setSamples() [1/2] void QwtPlotHistogram::setSamples (const QVector< QwtIntervalSample > & *samples*)

Initialize data with an array of samples.

Parameters

samples	Vector of points

Definition at line 299 of file qwt_plot_histogram.cpp.

14.82.4.18 setSamples() [2/2] void QwtPlotHistogram::setSamples (QwtSeriesData< QwtIntervalSample > * *data*)

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 315 of file qwt_plot_histogram.cpp.

14.82.4.19 setStyle() void QwtPlotHistogram::setStyle (HistogramStyle *style*)

Set the histogram's drawing style

Parameters

style	Histogram style

See also

HistogramStyle, style()

Definition at line 104 of file qwt_plot_histogram.cpp.

14.82.4.20 setSymbol() `void QwtPlotHistogram::setSymbol ( const QwtColumnSymbol * symbol )`

Assign a symbol.

In Column style an optional symbol can be assigned, that is responsible for displaying the rectangle that is defined by the interval and the distance between baseline() and value. When no symbol has been defined the area is displayed as plain rectangle using pen() and brush().

See also

style(), symbol(), drawColumn(), pen(), brush()

Note

In applications, where different intervals need to be displayed in a different way (f.e different colors or even using different symbols) it is recommended to overload drawColumn().

Definition at line 208 of file qwt_plot_histogram.cpp.

14.82.4.21 style() `QwtPlotHistogram::HistogramStyle QwtPlotHistogram::style ( ) const`

Returns

Style of the histogram

See also

HistogramStyle, setStyle()

Definition at line 119 of file qwt_plot_histogram.cpp.

14.82.4.22 symbol() `const QwtColumnSymbol * QwtPlotHistogram::symbol ( ) const`

Returns

Current symbol or NULL, when no symbol has been assigned

See also

setSymbol()

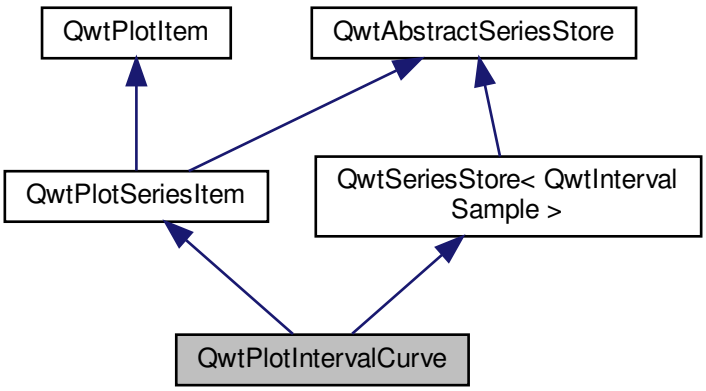
Definition at line 224 of file qwt_plot_histogram.cpp.

14.83 QwtPlotIntervalCurve Class Reference

QwtPlotIntervalCurve represents a series of samples, where each value is associated with an interval ($[y1, y2] = f(x)$).

```
#include <qwt_plot_intervalcurve.h>
```

Inheritance diagram for QwtPlotIntervalCurve:



Public Types

- enum CurveStyle { NoCurve , Tube , UserCurve = 100 }

  Curve styles. The default setting is QwtPlotIntervalCurve::Tube.
- enum PaintAttribute { ClipPolygons = 0x01 , ClipSymbol = 0x02 }
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPlotIntervalCurve (const QString &title=QString())
- QwtPlotIntervalCurve (const QwtText &title)
- virtual ~QwtPlotIntervalCurve ()

  Destructor.
- virtual int rtti () const override
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setSamples (const QVector< QwtIntervalSample > &)
- void setSamples (QwtSeriesData< QwtIntervalSample > *)
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)

  Assign a pen.
- const QPen & pen () const
- void setBrush (const QBrush &)
- const QBrush & brush () const

- void setStyle (CurveStyle style)
- CurveStyle style () const
- void setSymbol (const QwtIntervalSymbol *)
- const QwtIntervalSymbol * symbol () const
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual QRectF boundingRect () const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override

Protected Member Functions

- void init ()

  Initialize internal members.
- virtual void drawTube (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void drawSymbols (QPainter *, const QwtIntervalSymbol &, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const

14.83.1 Detailed Description

QwtPlotIntervalCurve represents a series of samples, where each value is associated with an interval ($[y1, y2] = f(x)$).

The representation depends on the style() and an optional symbol() that is displayed for each interval. QwtPlotIntervalCurve might be used to display error bars or the area between 2 curves.

Definition at line 27 of file qwt_plot_intervalcurve.h.

14.83.2 Member Typedef Documentation

14.83.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotIntervalCurve::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 77 of file qwt_plot_intervalcurve.h.

14.83.3 Member Enumeration Documentation

14.83.3.1 CurveStyle `enum QwtPlotIntervalCurve::CurveStyle`

Curve styles. The default setting is QwtPlotIntervalCurve::Tube.

See also

setStyle(), style()

Enumerator

NoCurve	Don't draw a curve. Note: This doesn't affect the symbols.
Tube	Build 2 curves from the upper and lower limits of the intervals and draw them with the pen(). The area between the curves is filled with the brush().
UserCurve	Styles >= QwtPlotIntervalCurve::UserCurve are reserved for derived classes that overload drawSeries() with additional application specific curve types.

Definition at line 38 of file qwt_plot_intervalcurve.h.

14.83.3.2 PaintAttribute `enum QwtPlotIntervalCurve::PaintAttribute`

Attributes to modify the drawing algorithm.

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

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.
ClipSymbol	Check if a symbol is on the plot canvas before painting it.

Definition at line 64 of file qwt_plot_intervalcurve.h.

14.83.4 Constructor & Destructor Documentation

14.83.4.1 QwtPlotIntervalCurve() [1/2]

```
QwtPlotIntervalCurve::QwtPlotIntervalCurve (
        const QString & title = QString() )  [explicit]
```

Constructor

Parameters

title	Title of the curve

Definition at line 90 of file qwt_plot_intervalcurve.cpp.

14.83.4.2 QwtPlotIntervalCurve() [2/2]

```
QwtPlotIntervalCurve::QwtPlotIntervalCurve (
        const QwtText & title )  [explicit]
```

Constructor

Parameters

title	Title of the curve

Definition at line 80 of file qwt_plot_intervalcurve.cpp.

14.83.5 Member Function Documentation

14.83.5.1 boundingRect() `QRectF QwtPlotIntervalCurve::boundingRect ( ) const [override], [virtual]`

Returns

Bounding rectangle of all samples. For an empty series the rectangle is invalid.

Reimplemented from QwtPlotSeriesItem.

Definition at line 300 of file qwt_plot_intervalcurve.cpp.

14.83.5.2 brush() `const QBrush & QwtPlotIntervalCurve::brush ( ) const`

Returns

Brush used to fill the area in Tube style()

See also

setBrush(), setStyle(), CurveStyle

Definition at line 291 of file qwt_plot_intervalcurve.cpp.

14.83.5.3 drawSeries()

```
void QwtPlotIntervalCurve::drawSeries (
        QPainter * painter,
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect,
        int from,
        int to ) const [override], [virtual]
```

Draw a subset of the samples

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to $<$ 0 the series will be painted to its last sample.

See also

drawTube(), drawSymbols()

Implements QwtPlotSeriesItem.

Definition at line 322 of file qwt_plot_intervalcurve.cpp.

14.83.5.4 drawSymbols()

```
void QwtPlotIntervalCurve::drawSymbols (
    QPainter * painter,
    const QwtIntervalSymbol & symbol,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw symbols for a subset of the samples

Parameters

painter	Painter
symbol	Interval symbol
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted

See also

setSymbol(), drawSeries(), drawTube()

Definition at line 487 of file qwt_plot_intervalcurve.cpp.

14.83.5.5 drawTube()

```
void QwtPlotIntervalCurve::drawTube (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw a tube

Builds 2 curves from the upper and lower limits of the intervals and draws them with the pen(). The area between the curves is filled with the brush().

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the series will be painted to its last sample.

See also

drawSeries(), drawSymbols()

Definition at line 371 of file qwt_plot_intervalcurve.cpp.

14.83.5.6 legendIcon()

```
QwtGraphic QwtPlotIntervalCurve::legendIcon (
    int index,
    const QSizeF & size ) const  [override], [virtual]
```

Returns

Icon for the legend

In case of Tube style() the icon is a plain rectangle filled with the brush(). If a symbol is assigned it is scaled to size.

Parameters

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

See also

QwtPlotItem::setLegendIconSize(), QwtPlotItem::legendData()

Reimplemented from QwtPlotItem.

Definition at line 554 of file qwt_plot_intervalcurve.cpp.

14.83.5.7 pen() `const QPen & QwtPlotIntervalCurve::pen ( ) const`

Returns

Pen used to draw the lines

See also

setPen(), brush()

Definition at line 263 of file qwt_plot_intervalcurve.cpp.

14.83.5.8 rtti() `int QwtPlotIntervalCurve::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotIntervalCurve

Reimplemented from QwtPlotItem.

Definition at line 115 of file qwt_plot_intervalcurve.cpp.

14.83.5.9 setBrush() `void QwtPlotIntervalCurve::setBrush ( const QBrush & brush )`

Assign a brush.

The brush is used to fill the area in Tube style().

Parameters

brush	Brush

See also

brush(), pen(), setStyle(), CurveStyle

Definition at line 276 of file qwt_plot_intervalcurve.cpp.

14.83.5.10 setPaintAttribute() `void QwtPlotIntervalCurve::setPaintAttribute ( PaintAttribute attribute, bool on = true )`

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 127 of file qwt_plot_intervalcurve.cpp.

14.83.5.11 setPen() [1/2]

```
void QwtPlotIntervalCurve::setPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 238 of file qwt_plot_intervalcurve.cpp.

14.83.5.12 setPen() [2/2]

```
void QwtPlotIntervalCurve::setPen (
    const QPen & pen )
```

Assign a pen.

Parameters

pen	New pen

See also

pen(), brush()

Definition at line 248 of file qwt_plot_intervalcurve.cpp.

14.83.5.13 setSamples() [1/2] void QwtPlotIntervalCurve::setSamples (const QVector< QwtIntervalSample > & *samples*)

Initialize data with an array of samples.

Parameters

samples	Vector of samples

Definition at line 150 of file qwt_plot_intervalcurve.cpp.

14.83.5.14 setSamples() [2/2] void QwtPlotIntervalCurve::setSamples (QwtSeriesData< QwtIntervalSample > * *data*)

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 166 of file qwt_plot_intervalcurve.cpp.

14.83.5.15 setStyle() void QwtPlotIntervalCurve::setStyle (CurveStyle *style*)

Set the curve's drawing style

Parameters

style	Curve style

See also

CurveStyle, style()

Definition at line 178 of file qwt_plot_intervalcurve.cpp.

14.83.5.16 setSymbol() void QwtPlotIntervalCurve::setSymbol (const QwtIntervalSymbol * *symbol*)

Assign a symbol.

Parameters

symbol	Symbol

See also

symbol()

Definition at line 204 of file qwt_plot_intervalcurve.cpp.

14.83.5.17 style() QwtPlotIntervalCurve::CurveStyle QwtPlotIntervalCurve::style ( ) const

Returns

Style of the curve

See also

setStyle()

Definition at line 193 of file qwt_plot_intervalcurve.cpp.

14.83.5.18 symbol() const QwtIntervalSymbol * QwtPlotIntervalCurve::symbol ( ) const

Returns

Current symbol or NULL, when no symbol has been assigned

See also

setSymbol()

Definition at line 220 of file qwt_plot_intervalcurve.cpp.

14.83.5.19 testPaintAttribute() `bool QwtPlotIntervalCurve::testPaintAttribute ( PaintAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

PaintAttribute, setPaintAttribute()

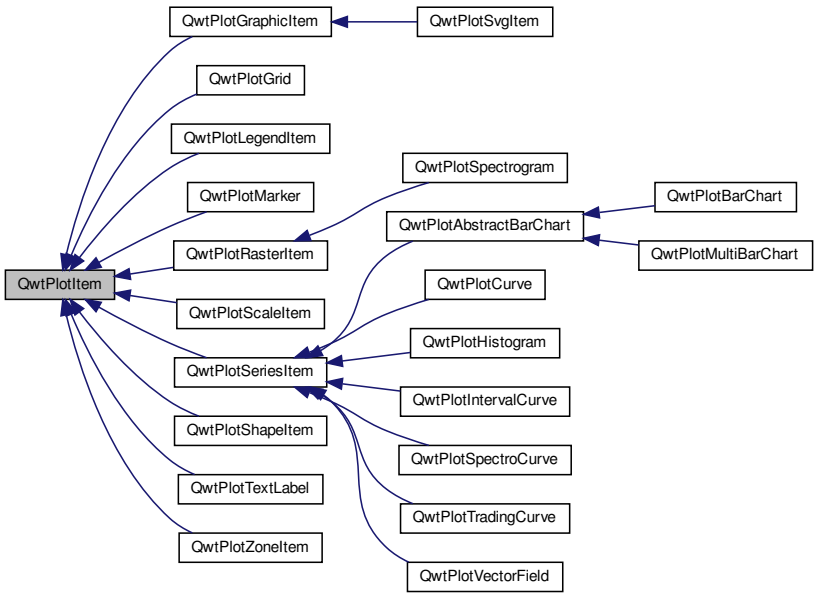
Definition at line 140 of file qwt_plot_intervalcurve.cpp.

14.84 QwtPlotItem Class Reference

Base class for items on the plot canvas.

`#include <qwt_plot_item.h>`

Inheritance diagram for QwtPlotItem:



Public Types

- enum RttiValues {
  Rtti_PlotItem = 0 , Rtti_PlotGrid , Rtti_PlotScale , Rtti_PlotLegend ,
  Rtti_PlotMarker , Rtti_PlotCurve , Rtti_PlotSpectroCurve , Rtti_PlotIntervalCurve ,
  Rtti_PlotHistogram , Rtti_PlotSpectrogram , Rtti_PlotGraphic , Rtti_PlotTradingCurve ,
  Rtti_PlotBarChart , Rtti_PlotMultiBarChart , Rtti_PlotShape , Rtti_PlotTextLabel ,
  Rtti_PlotZone , Rtti_PlotVectorField , Rtti_PlotUserItem = 1000 }

Runtime type information.

- enum ItemAttribute { Legend = 0x01 , AutoScale = 0x02 , Margins = 0x04 }

  Plot Item Attributes.
- enum ItemInterest { ScaleInterest = 0x01 , LegendInterest = 0x02 }

  Plot Item Interests.
- enum RenderHint { RenderAntialiased = 0x1 }

  Render hints.
- typedef QFlags< ItemAttribute > ItemAttributes
- typedef QFlags< ItemInterest > ItemInterests
- typedef QFlags< RenderHint > RenderHints

Public Member Functions

- QwtPlotItem ()
- QwtPlotItem (const QString &title)
- QwtPlotItem (const QwtText &title)
- virtual ~QwtPlotItem ()

  Destroy the QwtPlotItem.
- void attach (QwtPlot *plot)

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

  This method detaches a QwtPlotItem from any QwtPlot it has been associated with.
- QwtPlot * plot () const

  Return attached plot.
- void setTitle (const QString &title)
- void setTitle (const QwtText &title)
- const QwtText & title () const
- virtual int rtti () const
- void setItemAttribute (ItemAttribute, bool on=true)
- bool testItemAttribute (ItemAttribute) const
- void setItemInterest (ItemInterest, bool on=true)
- bool testItemInterest (ItemInterest) const
- void setRenderHint (RenderHint, bool on=true)
- bool testRenderHint (RenderHint) const
- void setRenderThreadCount (uint numThreads)
- uint renderThreadCount () const
- void setLegendIconSize (const QSize &)
- QSize legendIconSize () const
- double z () const
- void setZ (double z)

  Set the z value.
- void show ()

  Show the item.
- void hide ()

  Hide the item.
- virtual void setVisible (bool)
- bool isVisible () const
- void setAxes (QwtAxisId xAxis, QwtAxisId yAxis)
- void setXAxis (QwtAxisId)
- QwtAxisId xAxis () const

  Return xAxis.
- void setYAxis (QwtAxisId)

- QwtAxisId yAxis () const

  Return yAxis.
- virtual void itemChanged ()
- virtual void legendChanged ()
- virtual void draw (QPainter *painter, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const =0

  Draw the item.
- virtual QRectF boundingRect () const
- virtual void getCanvasMarginHint (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, double &left, double &top, double &right, double &bottom) const

  Calculate a hint for the canvas margin.
- virtual void updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &)

  Update the item to changes of the axes scale division.
- virtual void updateLegend (const QwtPlotItem *, const QList< QwtLegendData > &)

  Update the item to changes of the legend info.
- QRectF scaleRect (const QwtScaleMap &, const QwtScaleMap &) const

  Calculate the bounding scale rectangle of 2 maps.
- QRectF paintRect (const QwtScaleMap &, const QwtScaleMap &) const

  Calculate the bounding paint rectangle of 2 maps.
- virtual QList< QwtLegendData > legendData () const

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

Protected Member Functions

- QwtGraphic defaultIcon (const QBrush &, const QSizeF &) const

  Return a default icon from a brush.

14.84.1 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.

Definition at line 66 of file qwt_plot_item.h.

14.84.2 Member Typedef Documentation

14.84.2.1 ItemAttributes `typedef QFlags<ItemAttribute > QwtPlotItem::ItemAttributes`

An ORed combination of ItemAttribute values.

Definition at line 167 of file qwt_plot_item.h.

14.84.2.2 ItemInterests `typedef QFlags<ItemInterest > QwtPlotItem::ItemInterests`

An ORed combination of ItemInterest values.

Definition at line 200 of file qwt_plot_item.h.

14.84.2.3 RenderHints `typedef QFlags<RenderHint > QwtPlotItem::RenderHints`

An ORed combination of RenderHint values.

Definition at line 209 of file qwt_plot_item.h.

14.84.3 Member Enumeration Documentation

14.84.3.1 ItemAttribute `enum QwtPlotItem::ItemAttribute`

Plot Item Attributes.

Various aspects of a plot widget depend on the attributes of the attached plot items. If and how a single plot item participates in these updates depends on its attributes.

See also

setItemAttribute(), testItemAttribute(), ItemInterest

Enumerator

Legend	The item is represented on the legend.
AutoScale	The boundingRect() of the item is included in the autoscaling calculation as long as its width or height is >= 0.0.
Margins	The item needs extra space to display something outside its bounding rectangle. **See also** getCanvasMarginHint()

Definition at line 147 of file qwt_plot_item.h.

14.84.3.2 ItemInterest `enum QwtPlotItem::ItemInterest`

Plot Item Interests.

Plot items might depend on the situation of the corresponding plot widget. By enabling an interest the plot item will be notified, when the corresponding attribute of the plot widgets has changed.

See also

setItemAttribute(), testItemAttribute(), ItemInterest

Enumerator

ScaleInterest	The item is interested in updates of the scales **See also** updateScaleDiv()
LegendInterest	The item is interested in updates of the legend (of other items) This flag is intended for items, that want to implement a legend for displaying entries of other plot item. **Note** If the plot item wants to be represented on a legend enable QwtPlotItem::Legend instead. **See also** updateLegend()

Definition at line 179 of file qwt_plot_item.h.

14.84.3.3 RenderHint `enum QwtPlotItem::RenderHint`

Render hints.

Enumerator

RenderAntialiased	Enable antialiasing.

Definition at line 203 of file qwt_plot_item.h.

14.84.3.4 RttiValues `enum QwtPlotItem::RttiValues`

Runtime type information.

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

Enumerator

Rtti_PlotItem	Unspecific value, that can be used, when it doesn't matter.
Rtti_PlotGrid	For QwtPlotGrid.
Rtti_PlotScale	For QwtPlotScaleItem.
Rtti_PlotLegend	For QwtPlotLegendItem.
Rtti_PlotMarker	For QwtPlotMarker.
Rtti_PlotCurve	For QwtPlotCurve.
Rtti_PlotSpectroCurve	For QwtPlotSpectroCurve.
Rtti_PlotIntervalCurve	For QwtPlotIntervalCurve.
Rtti_PlotHistogram	For QwtPlotHistogram.
Rtti_PlotSpectrogram	For QwtPlotSpectrogram.
Rtti_PlotGraphic	For QwtPlotGraphicItem, QwtPlotSvgItem.
Rtti_PlotTradingCurve	For QwtPlotTradingCurve.
Rtti_PlotBarChart	For QwtPlotBarChart.
Rtti_PlotMultiBarChart	For QwtPlotMultiBarChart.
Rtti_PlotShape	For QwtPlotShapeItem.
Rtti_PlotTextLabel	For QwtPlotTextLabel.
Rtti_PlotZone	For QwtPlotZoneItem.
Rtti_PlotVectorField	For QwtPlotVectorField.
Rtti_PlotUserItem	Values >= Rtti_PlotUserItem are reserved for plot items not implemented in the Qwt library.

Definition at line 75 of file qwt_plot_item.h.

14.84.4 Constructor & Destructor Documentation

14.84.4.1 QwtPlotItem() [1/3] `QwtPlotItem::QwtPlotItem ( ) [explicit]`

Constructor

Definition at line 55 of file qwt_plot_item.cpp.

14.84.4.2 QwtPlotItem() [2/3] QwtPlotItem::QwtPlotItem (const QString & *title*) [explicit]

Constructor

Parameters

title	Title of the item

Definition at line 64 of file qwt_plot_item.cpp.

14.84.4.3 QwtPlotItem() [3/3] QwtPlotItem::QwtPlotItem (const QwtText & *title*) [explicit]

Constructor

Parameters

title	Title of the item

Definition at line 74 of file qwt_plot_item.cpp.

14.84.5 Member Function Documentation

14.84.5.1 attach() 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

plot	Plot widget

See also

detach()

Definition at line 98 of file qwt_plot_item.cpp.

14.84.5.2 boundingRect() `QRectF QwtPlotItem::boundingRect ( ) const [virtual]`

Returns

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

Note

A width or height < 0.0 is ignored by the autoscaler

Reimplemented in QwtPlotZoneItem, QwtPlotVectorField, QwtPlotTradingCurve, QwtPlotShapeItem, QwtPlotSeriesItem, QwtPlotRasterItem, QwtPlotMultiBarChart, QwtPlotMarker, QwtPlotIntervalCurve, QwtPlotHistogram, QwtPlotGraphicItem, and QwtPlotBarChart.

Definition at line 568 of file qwt_plot_item.cpp.

14.84.5.3 defaultIcon() `QwtGraphic QwtPlotItem::defaultIcon ( const QBrush & brush, const QSizeF & size ) const [protected]`

Return a default icon from a brush.

The default icon is a filled rectangle used in several derived classes as legendIcon().

Parameters

brush	Fill brush
size	Icon size

Returns

A filled rectangle

Definition at line 422 of file qwt_plot_item.cpp.

14.84.5.4 detach() `void QwtPlotItem::detach ( )`

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

detach() is equivalent to calling attach(NULL)

See also

attach()

Definition at line 119 of file qwt_plot_item.cpp.

14.84.5.5 draw()

```
virtual void QwtPlotItem::draw (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const  [pure virtual]
```

Draw the item.

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rect of the canvas in painter coordinates

Implemented in QwtPlotSpectrogram, QwtPlotShapeItem, QwtPlotSeriesItem, QwtPlotScaleItem, QwtPlotRasterItem, QwtPlotLegendItem, QwtPlotGrid, QwtPlotGraphicItem, QwtPlotMarker, QwtPlotZoneItem, and QwtPlotTextLabel.

14.84.5.6 getCanvasMarginHint()

```
void QwtPlotItem::getCanvasMarginHint (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    double & left,
    double & top,
    double & right,
    double & bottom ) const  [virtual]
```

Calculate a hint for the canvas margin.

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

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

Parameters

xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas in painter coordinates
left	Returns the left margin
top	Returns the top margin
right	Returns the right margin
bottom	Returns the bottom margin

The default implementation returns 0 for all margins

See also

QwtPlot::getCanvasMarginsHint(), QwtPlot::updateCanvasMargins()

Reimplemented in QwtPlotAbstractBarChart.

Definition at line 595 of file qwt_plot_item.cpp.

14.84.5.7 isVisible() `bool QwtPlotItem::isVisible ( ) const`

Returns

true if visible

See also

setVisible(), show(), hide()

Definition at line 470 of file qwt_plot_item.cpp.

14.84.5.8 itemChanged() `void QwtPlotItem::itemChanged ( ) [virtual]`

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

See also

QwtPlot::legendChanged(), QwtPlot::autoRefresh()

Definition at line 481 of file qwt_plot_item.cpp.

14.84.5.9 legendChanged() `void QwtPlotItem::legendChanged ( ) [virtual]`

Update the legend of the parent plot.

See also

QwtPlot::updateLegend(), itemChanged()

Definition at line 491 of file qwt_plot_item.cpp.

14.84.5.10 legendData() QList< QwtLegendData > QwtPlotItem::legendData ( ) const [virtual]

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

Most items are represented by one entry on the legend showing an icon and a text, but f.e. QwtPlotMultiBarChart displays one entry for each bar.

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

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

Returns

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

See also

title(), legendIcon(), QwtLegend, QwtPlotLegendItem

Reimplemented in QwtPlotMultiBarChart, and QwtPlotBarChart.

Definition at line 626 of file qwt_plot_item.cpp.

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

Returns

Icon representing the item on the legend

The default implementation returns an invalid icon

Parameters

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

See also

setLegendIconSize(), legendData()

Reimplemented in QwtPlotVectorField, QwtPlotTradingCurve, QwtPlotShapeItem, QwtPlotMultiBarChart, QwtPlotMarker, QwtPlotIntervalCurve, QwtPlotHistogram, QwtPlotCurve, and QwtPlotBarChart.

Definition at line 402 of file qwt_plot_item.cpp.

14.84.5.12 legendIconSize() `QSize QwtPlotItem::legendIconSize ( ) const`

Returns

Legend icon size

See also

setLegendIconSize(), legendIcon()

Definition at line 386 of file qwt_plot_item.cpp.

14.84.5.13 paintRect() `QRectF QwtPlotItem::paintRect ( const QwtScaleMap & xMap, const QwtScaleMap & yMap ) const`

Calculate the bounding paint rectangle of 2 maps.

Parameters

xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.

Returns

Bounding paint rectangle of the scale maps, not normalized

Definition at line 720 of file qwt_plot_item.cpp.

14.84.5.14 renderThreadCount() `uint QwtPlotItem::renderThreadCount ( ) const`

Returns

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

Definition at line 360 of file qwt_plot_item.cpp.

14.84.5.15 rtti() `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 QwtPlotZoneItem, QwtPlotVectorField, QwtPlotTradingCurve, QwtPlotTextLabel, QwtPlotSpectrogram, QwtPlotSpectroCurve, QwtPlotShapeItem, QwtPlotScaleItem, QwtPlotMultiBarChart, QwtPlotMarker, QwtPlotLegendItem, QwtPlotIntervalCurve, QwtPlotHistogram, QwtPlotGrid, QwtPlotGraphicItem, QwtPlotCurve, and QwtPlotBarChart.

Definition at line 136 of file qwt_plot_item.cpp.

14.84.5.16 scaleRect() `QRectF QwtPlotItem::scaleRect ( const QwtScaleMap & xMap, const QwtScaleMap & yMap ) const`

Calculate the bounding scale rectangle of 2 maps.

Parameters

xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.

Returns

Bounding scale rect of the scale maps, not normalized

Definition at line 705 of file qwt_plot_item.cpp.

14.84.5.17 setAxes() `void QwtPlotItem::setAxes ( QwtAxisId xAxisId, QwtAxisId yAxisId )`

Set X and Y axis

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

Parameters

x↩ AxisId	X Axis
y↩ AxisId	Y Axis

See also

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

Definition at line 507 of file qwt_plot_item.cpp.

14.84.5.18 setItemAttribute()

```
void QwtPlotItem::setItemAttribute (
    ItemAttribute attribute,
    bool on = true )
```

Toggle an item attribute

Parameters

attribute	Attribute type
on	true/false

See also

testItemAttribute(), ItemInterest

Definition at line 228 of file qwt_plot_item.cpp.

14.84.5.19 setItemInterest()

```
void QwtPlotItem::setItemInterest (
    ItemInterest interest,
    bool on = true )
```

Toggle an item interest

Parameters

interest	Interest type
on	true/false

See also

testItemInterest(), ItemAttribute

Definition at line 279 of file qwt_plot_item.cpp.

14.84.5.20 setLegendIconSize() `void QwtPlotItem::setLegendIconSize ( const QSize & size )`

Set the size of the legend icon

The default setting is 8x8 pixels

Parameters

size	Size

See also

legendIconSize(), legendIcon()

Definition at line 373 of file qwt_plot_item.cpp.

14.84.5.21 setRenderHint() `void QwtPlotItem::setRenderHint ( RenderHint hint, bool on = true )`

Toggle an render hint

Parameters

hint	Render hint
on	true/false

See also

testRenderHint(), RenderHint

Definition at line 312 of file qwt_plot_item.cpp.

14.84.5.22 setRenderThreadCount() `void QwtPlotItem::setRenderThreadCount ( uint numThreads )`

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

The default setting is set to 1.

Parameters

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

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

Definition at line 350 of file qwt_plot_item.cpp.

14.84.5.23 setTitle() [1/2] void QwtPlotItem::setTitle (const QString & *title*)

Set a new title

Parameters

title	Title

See also

title()

Definition at line 187 of file qwt_plot_item.cpp.

14.84.5.24 setTitle() [2/2] void QwtPlotItem::setTitle (const QwtText & *title*)

Set a new title

Parameters

title	Title

See also

title()

Definition at line 198 of file qwt_plot_item.cpp.

14.84.5.25 setVisible() void QwtPlotItem::setVisible (bool *on*) [virtual]

Show/Hide the item

Parameters

on	Show if true, otherwise hide

See also

isVisible(), show(), hide()

Definition at line 457 of file qwt_plot_item.cpp.

14.84.5.26 setXAxis() `void QwtPlotItem::setXAxis ( QwtAxisId axisId )`

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

axis↩ Id	X Axis

See also

setAxes(), setYAxis(), xAxis()

Definition at line 526 of file qwt_plot_item.cpp.

14.84.5.27 setYAxis() `void QwtPlotItem::setYAxis ( QwtAxisId axisId )`

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

axis↩ Id	Y Axis

See also

setAxes(), setXAxis(), yAxis()

Definition at line 543 of file qwt_plot_item.cpp.

14.84.5.28 setZ() void QwtPlotItem::setZ (double z)

Set the z value.

Plot items are painted in increasing z-order.

Parameters

z	Z-value

See also

z(), QwtPlotDict::itemList()

Definition at line 165 of file qwt_plot_item.cpp.

14.84.5.29 testItemAttribute() bool QwtPlotItem::testItemAttribute (ItemAttribute attribute) const

Test an item attribute

Parameters

attribute	Attribute type

Returns

true/false

See also

setItemAttribute(), ItemInterest

Definition at line 266 of file qwt_plot_item.cpp.

14.84.5.30 testItemInterest() bool QwtPlotItem::testItemInterest (ItemInterest interest) const

Test an item interest

Parameters

interest	Interest type

Returns

true/false

See also

setItemInterest(), ItemAttribute

Definition at line 299 of file qwt_plot_item.cpp.

14.84.5.31 testRenderHint() `bool QwtPlotItem::testRenderHint ( RenderHint hint ) const`

Test a render hint

Parameters

hint	Render hint

Returns

true/false

See also

setRenderHint(), RenderHint

Definition at line 332 of file qwt_plot_item.cpp.

14.84.5.32 title() `const QwtText & QwtPlotItem::title ( ) const`

Returns

Title of the item

See also

setTitle()

Definition at line 215 of file qwt_plot_item.cpp.

14.84.5.33 updateLegend() `void QwtPlotItem::updateLegend ( const QwtPlotItem * item, const QList< QwtLegendData > & data ) [virtual]`

Update the item to changes of the legend info.

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

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

Parameters

item	Plot item to be displayed on a legend
data	Attributes how to display item on the legend

See also

QwtPlotLegendItem

Note

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

Reimplemented in QwtPlotLegendItem.

Definition at line 690 of file qwt_plot_item.cpp.

14.84.5.34 updateScaleDiv()

```
void QwtPlotItem::updateScaleDiv (
    const QwtScaleDiv & xScaleDiv,
    const QwtScaleDiv & yScaleDiv ) [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()

updateScaleDiv() is only called when the ScaleInterest interest is enabled. The default implementation does nothing.

Parameters

xScaleDiv	Scale division of the x-axis
yScaleDiv	Scale division of the y-axis

See also

QwtPlot::updateAxes(), ScaleInterest

Reimplemented in QwtPlotGrid, QwtPlotSeriesItem, and QwtPlotScaleItem.

Definition at line 665 of file qwt_plot_item.cpp.

14.84.5.35 z()

```
double QwtPlotItem::z ( ) const
```

Plot items are painted in increasing z-order.

Returns

setZ(), QwtPlotDict::itemList()

Definition at line 152 of file qwt_plot_item.cpp.

14.85 QwtPlotLayout Class Reference

Layout engine for QwtPlot.

`#include <qwt_plot_layout.h>`

Public Types

- enum Option {
  AlignScales = 0x01 , IgnoreScrollbars = 0x02 , IgnoreFrames = 0x04 , IgnoreLegend = 0x08 ,
  IgnoreTitle = 0x10 , IgnoreFooter = 0x20 }
- typedef QFlags< Option > Options

Public Member Functions

- QwtPlotLayout ()

  Constructor.
- virtual ~QwtPlotLayout ()

  Destructor.
- void setCanvasMargin (int margin, int axis=-1)
- int canvasMargin (int axisId) const
- void setAlignCanvasToScales (bool)

  Set the align-canvas-to-axis-scales flag for all axes.
- void setAlignCanvasToScale (int axisId, bool)
- bool alignCanvasToScale (int axisId) const
- void setSpacing (int)
- int spacing () const
- void setLegendPosition (QwtPlot::LegendPosition pos, double ratio)

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

  Specify the position of the legend.
- QwtPlot::LegendPosition legendPosition () const
- void setLegendRatio (double ratio)
- double legendRatio () const
- virtual QSize minimumSizeHint (const QwtPlot *) const
- virtual void activate (const QwtPlot *, const QRectF &plotRect, Options options=Options())

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

Protected Member Functions

- void setTitleRect (const QRectF &)

  Set the geometry for the title.
- void setFooterRect (const QRectF &)

  Set the geometry for the footer.
- void setLegendRect (const QRectF &)

  Set the geometry for the legend.
- void setScaleRect (QwtAxisId, const QRectF &)

  Set the geometry for an axis.
- void setCanvasRect (const QRectF &)

  Set the geometry for the canvas.

14.85.1 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.

See also

QwtPlot::setPlotLayout()

Definition at line 27 of file qwt_plot_layout.h.

14.85.2 Member Typedef Documentation

14.85.2.1 Options `typedef QFlags<Option > QwtPlotLayout::Options`

An ORed combination of Option values.

Definition at line 58 of file qwt_plot_layout.h.

14.85.3 Member Enumeration Documentation

14.85.3.1 Option `enum QwtPlotLayout::Option`

Options to configure the plot layout engine

See also

activate(), QwtPlotRenderer

Enumerator

AlignScales	Unused.
IgnoreScrollbars	Ignore the dimension of the scrollbars. There are no scrollbars, when the plot is not rendered to widgets.
IgnoreFrames	Ignore all frames.
IgnoreLegend	Ignore the legend.
IgnoreTitle	Ignore the title.
IgnoreFooter	Ignore the footer.

Definition at line 34 of file qwt_plot_layout.h.

14.85.4 Member Function Documentation

14.85.4.1 activate()

```
void QwtPlotLayout::activate (
    const QwtPlot * plot,
    const QRectF & plotRect,
    Options options = Options() )  [virtual]
```

Recalculate the geometry of all components.

Parameters

plot	Plot to be layout
plotRect	Rectangle where to place the components
options	Layout options

See also

invalidate(), titleRect(), footerRect() legendRect(), scaleRect(), canvasRect()

Definition at line 1508 of file qwt_plot_layout.cpp.

14.85.4.2 alignCanvasToScale()

```
bool QwtPlotLayout::alignCanvasToScale (
    int axisPos ) 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.

Parameters

axisPos	Axis position

Returns

align-canvas-to-axis-scales setting

See also

setAlignCanvasToScale(), setAlignCanvasToScale(), setCanvasMargin()

Definition at line 1141 of file qwt_plot_layout.cpp.

14.85.4.3 canvasMargin()

```
int QwtPlotLayout::canvasMargin (
    int axisPos ) const
```

Parameters

axisPos	Axis position

Returns

Margin around the scale tick borders

See also

setCanvasMargin()

Definition at line 1089 of file qwt_plot_layout.cpp.

14.85.4.4 canvasRect() `QRectF QwtPlotLayout::canvasRect ( ) const`

Returns

Geometry for the canvas

See also

activate(), invalidate()

Definition at line 1380 of file qwt_plot_layout.cpp.

14.85.4.5 footerRect() `QRectF QwtPlotLayout::footerRect ( ) const`

Returns

Geometry for the footer

See also

activate(), invalidate()

Definition at line 1304 of file qwt_plot_layout.cpp.

14.85.4.6 invalidate() `void QwtPlotLayout::invalidate ( ) [virtual]`

Invalidate the geometry of all components.

See also

activate()

Definition at line 1389 of file qwt_plot_layout.cpp.

14.85.4.7 legendPosition() `QwtPlot::LegendPosition QwtPlotLayout::legendPosition ( ) const`

Returns

Position of the legend

See also

setLegendPosition(), QwtPlot::setLegendPosition(), QwtPlot::legendPosition()

Definition at line 1237 of file qwt_plot_layout.cpp.

14.85.4.8 legendRatio() `double QwtPlotLayout::legendRatio ( ) const`

Returns

The relative size of the legend in the plot.

See also

setLegendPosition()

Definition at line 1260 of file qwt_plot_layout.cpp.

14.85.4.9 legendRect() `QRectF QwtPlotLayout::legendRect ( ) const`

Returns

Geometry for the legend

See also

activate(), invalidate()

Definition at line 1328 of file qwt_plot_layout.cpp.

14.85.4.10 minimumSizeHint() `QSize QwtPlotLayout::minimumSizeHint ( const QwtPlot * plot ) const [virtual]`

Returns

Minimum size hint

Parameters

plot	Plot widget

See also

QwtPlot::minimumSizeHint()

Definition at line 1404 of file qwt_plot_layout.cpp.

14.85.4.11 scaleRect() `QRectF QwtPlotLayout::scaleRect ( QwtAxisId axisId ) const`

Parameters

axisId	Axis

Returns

Geometry for the scale

See also

activate(), invalidate()

Definition at line 1355 of file qwt_plot_layout.cpp.

14.85.4.12 setAlignCanvasToScale() `void QwtPlotLayout::setAlignCanvasToScale ( int axisPos, bool on )`

Change the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size,
- align with the axis scale ends to control its size.

The axisId parameter is somehow confusing as it identifies a border of the plot and not the axes, that are aligned. F.e when QwtAxis::YLeft is set, the left end of the the x-axes (QwtAxis::XTop, QwtAxis::XBottom) is aligned.

Parameters

axisId	Axis index
on	New align-canvas-to-axis-scales setting

See also

setCanvasMargin(), alignCanvasToScale(), setAlignCanvasToScales()

Warning

In case of on == true canvasMargin() will have no effect

Definition at line 1126 of file qwt_plot_layout.cpp.

14.85.4.13 setAlignCanvasToScales()

```
void QwtPlotLayout::setAlignCanvasToScales (
    bool on )
```

Set the align-canvas-to-axis-scales flag for all axes.

Parameters

on	True/False

See also

setAlignCanvasToScale(), alignCanvasToScale()

Definition at line 1103 of file qwt_plot_layout.cpp.

14.85.4.14 setCanvasMargin()

```
void QwtPlotLayout::setCanvasMargin (
    int margin,
    int axisPos = -1 )
```

Change a margin of the canvas. The margin is the space above/below the scale ticks. A negative margin will be set to -1, excluding the borders of the scales.

Parameters

margin	New margin
axisPos	One of QwtAxis::Position. Specifies where the position of the margin. -1 means margin at all borders.

See also

canvasMargin()

Warning

The margin will have no effect when alignCanvasToScale() is true

Definition at line 1066 of file qwt_plot_layout.cpp.

14.85.4.15 setCanvasRect() `void QwtPlotLayout::setCanvasRect ( const QRectF & rect ) [protected]`

Set the geometry for the canvas.

This method is intended to be used from derived layouts overloading activate()

See also

canvasRect(), activate()

Definition at line 1371 of file qwt_plot_layout.cpp.

14.85.4.16 setFooterRect() `void QwtPlotLayout::setFooterRect ( const QRectF & rect ) [protected]`

Set the geometry for the footer.

This method is intended to be used from derived layouts overloading activate()

See also

footerRect(), activate()

Definition at line 1295 of file qwt_plot_layout.cpp.

14.85.4.17 setLegendPosition() [1/2] `void QwtPlotLayout::setLegendPosition ( QwtPlot::LegendPosition pos )`

Specify the position of the legend.

Parameters

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

See also

QwtPlot::setLegendPosition()

Definition at line 1227 of file qwt_plot_layout.cpp.

14.85.4.18 setLegendPosition() [2/2] `void QwtPlotLayout::setLegendPosition ( QwtPlot::LegendPosition pos, double ratio )`

Specify the position of the legend.

Parameters

pos	The legend's position.
ratio	Ratio between legend and the bounding rectangle of title, footer, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also

QwtPlot::setLegendPosition()

Definition at line 1183 of file qwt_plot_layout.cpp.

14.85.4.19 setLegendRatio() `void QwtPlotLayout::setLegendRatio ( double ratio )`

Specify the relative size of the legend in the plot

Parameters

ratio	Ratio between legend and the bounding rectangle of title, footer, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

Definition at line 1251 of file qwt_plot_layout.cpp.

14.85.4.20 setLegendRect() `void QwtPlotLayout::setLegendRect ( const QRectF & rect ) [protected]`

Set the geometry for the legend.

This method is intended to be used from derived layouts overloading activate()

Parameters

rect	Rectangle for the legend

See also

legendRect(), activate()

Definition at line 1319 of file qwt_plot_layout.cpp.

14.85.4.21 setScaleRect() `void QwtPlotLayout::setScaleRect ( QwtAxisId axisId, const QRectF & rect ) [protected]`

Set the geometry for an axis.

This method is intended to be used from derived layouts overloading activate()

Parameters

axisId	Axis
rect	Rectangle for the scale

See also

scaleRect(), activate()

Definition at line 1344 of file qwt_plot_layout.cpp.

14.85.4.22 setSpacing() `void QwtPlotLayout::setSpacing ( int spacing )`

Change the spacing of the plot. The spacing is the distance between the plot components.

Parameters

spacing	New spacing

See also

setCanvasMargin(), spacing()

Definition at line 1156 of file qwt_plot_layout.cpp.

14.85.4.23 setTitleRect() `void QwtPlotLayout::setTitleRect ( const QRectF & rect ) [protected]`

Set the geometry for the title.

This method is intended to be used from derived layouts overloading activate()

See also

titleRect(), activate()

Definition at line 1273 of file qwt_plot_layout.cpp.

14.85.4.24 spacing() `int QwtPlotLayout::spacing ( ) const`

Returns

Spacing

See also

margin(), setSpacing()

Definition at line 1165 of file qwt_plot_layout.cpp.

14.85.4.25 titleRect() `QRectF QwtPlotLayout::titleRect ( ) const`

Returns

Geometry for the title

See also

activate(), invalidate()

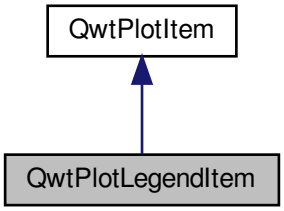
Definition at line 1282 of file qwt_plot_layout.cpp.

14.86 QwtPlotLegendItem Class Reference

A class which draws a legend inside the plot canvas.

`#include <qwt_plot_legenditem.h>`

Inheritance diagram for QwtPlotLegendItem:



Public Types

- enum BackgroundMode { LegendBackground , ItemBackground }

  Background mode.

Public Member Functions

- QwtPlotLegendItem ()

  Constructor.
- virtual ~QwtPlotLegendItem ()

  Destructor.
- virtual int rtti () const override
- void setAlignmentInCanvas (Qt::Alignment)

  Set the alignmnet.
- Qt::Alignment alignmentInCanvas () const
- void setOffsetInCanvas (Qt::Orientations, int numPixels)

  Set the distance between the legend and the canvas border.
- int offsetInCanvas (Qt::Orientation) const
- void setMaxColumns (uint)

  Limit the number of columns.
- uint maxColumns () const
- void setMargin (int)

  Set the margin around legend items.
- int margin () const
- void setSpacing (int)

  Set the spacing between the legend items.
- int spacing () const
- void setItemMargin (int)
- int itemMargin () const
- void setItemSpacing (int)
- int itemSpacing () const
- void setFont (const QFont &)
- QFont font () const
- void setBorderRadius (double)
- double borderRadius () const
- void setBorderPen (const QPen &)
- QPen borderPen () const
- void setBackgroundBrush (const QBrush &)

  Set the background brush.
- QBrush backgroundBrush () const
- void setBackgroundMode (BackgroundMode)

  Set the background mode.
- BackgroundMode backgroundMode () const
- void setTextPen (const QPen &)

  Set the pen for drawing text labels.
- QPen textPen () const
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override
- void clearLegend ()

  Remove all items from the legend.
- virtual void updateLegend (const QwtPlotItem *, const QList< QwtLegendData > &) override
- virtual QRect geometry (const QRectF &canvasRect) const
- virtual QSize minimumSize (const QwtLegendData &) const
- virtual int heightForWidth (const QwtLegendData &, int width) const
- QList< const QwtPlotItem * > plotItems () const
- QList< QRect > legendGeometries (const QwtPlotItem *) const

Protected Member Functions

- virtual void drawLegendData (QPainter *, const QwtPlotItem *, const QwtLegendData &, const QRectF &) const
- virtual void drawBackground (QPainter *, const QRectF &rect) const

14.86.1 Detailed Description

A class which draws a legend inside the plot canvas.

QwtPlotLegendItem can be used to draw a inside the plot canvas. It can be used together with a QwtLegend or instead of it to have more space for the plot canvas.

In opposite to QwtLegend the legend item is not interactive. To identify mouse clicks on a legend item an event filter needs to be installed catching mouse events ob the plot canvas. The geometries of the legend items are available using legendGeometries().

The legend item is aligned to plot canvas according to its alignment() flags. It might have a background for the complete legend (usually semi transparent) or for each legend item.

Note

An external QwtLegend with a transparent background on top the plot canvas might be another option with a similar effect.

Definition at line 41 of file qwt_plot_legenditem.h.

14.86.2 Member Enumeration Documentation

14.86.2.1 BackgroundMode `enum QwtPlotLegendItem::BackgroundMode`

Background mode.

Depending on the mode the complete legend or each item might have an background.

The default setting is LegendBackground.

See also

setBackgroundMode(), setBackgroundBrush(), drawBackground()

Enumerator

LegendBackground	The legend has a background.
ItemBackground	Each item has a background.

Definition at line 54 of file qwt_plot_legenditem.h.

14.86.3 Member Function Documentation

14.86.3.1 alignmentInCanvas() `Qt::Alignment QwtPlotLegendItem::alignmentInCanvas ( ) const`

Returns

Alignment flags

See also

setAlignmentInCanvas()

Definition at line 223 of file qwt_plot_legenditem.cpp.

14.86.3.2 backgroundBrush() `QBrush QwtPlotLegendItem::backgroundBrush ( ) const`

Returns

Brush is used to fill the background

See also

setBackgroundBrush(), backgroundMode(), drawBackground()

Definition at line 522 of file qwt_plot_legenditem.cpp.

14.86.3.3 backgroundMode() `QwtPlotLegendItem::BackgroundMode QwtPlotLegendItem::backgroundMode ( ) const`

Returns

backgroundMode

See also

setBackgroundMode(), backgroundBrush(), drawBackground()

Definition at line 550 of file qwt_plot_legenditem.cpp.

14.86.3.4 borderPen() `QPen QwtPlotLegendItem::borderPen ( ) const`

Returns

Pen for drawing the border

See also

setBorderPen(), backgroundBrush()

Definition at line 496 of file qwt_plot_legenditem.cpp.

14.86.3.5 borderRadius() `double QwtPlotLegendItem::borderRadius ( ) const`

Returns

Radius of the border

See also

setBorderRadius(), setBorderPen()

Definition at line 472 of file qwt_plot_legenditem.cpp.

14.86.3.6 draw()

```
void QwtPlotLegendItem::draw (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const  [override], [virtual]
```

Draw the legend

Parameters

painter	Painter
xMap	x Scale Map
yMap	y Scale Map
canvasRect	Contents rectangle of the canvas in painter coordinates

Implements QwtPlotItem.

Definition at line 587 of file qwt_plot_legenditem.cpp.

14.86.3.7 drawBackground() `void QwtPlotLegendItem::drawBackground ( QPainter * painter, const QRectF & rect ) const [protected], [virtual]`

Draw a rounded rect

Parameters

painter	Painter
rect	Bounding rectangle

See also

setBorderRadius(), setBorderPen(), setBackgroundBrush(), setBackgroundMode()

Definition at line 630 of file qwt_plot_legenditem.cpp.

14.86.3.8 drawLegendData() `void QwtPlotLegendItem::drawLegendData ( QPainter * painter, const QwtPlotItem * plotItem, const QwtLegendData & data, const QRectF & rect ) const [protected], [virtual]`

Draw an entry on the legend

Parameters

painter	Qt Painter
plotItem	Plot item, represented by the entry
data	Attributes of the legend entry
rect	Bounding rectangle for the entry

Definition at line 780 of file qwt_plot_legenditem.cpp.

14.86.3.9 font() `QFont QwtPlotLegendItem::font ( ) const`

Returns

Font used for drawing the text label

See also

setFont()

Definition at line 389 of file qwt_plot_legenditem.cpp.

14.86.3.10 geometry()

```
QRect QwtPlotLegendItem::geometry (
    const QRectF & canvasRect ) const  [virtual]
```

Calculate the geometry of the legend on the canvas

Parameters

canvasRect	Geometry of the canvas

Returns

Geometry of the legend

Definition at line 650 of file qwt_plot_legenditem.cpp.

14.86.3.11 heightForWidth()

```
int QwtPlotLegendItem::heightForWidth (
    const QwtLegendData & data,
    int width ) const  [virtual]
```

Returns

The preferred height, for a width.

Parameters

data	Attributes of the legend entry
width	Width

Definition at line 862 of file qwt_plot_legenditem.cpp.

14.86.3.12 itemMargin()

```
int QwtPlotLegendItem::itemMargin ( ) const
```

Returns

Margin around each item

See also

setItemMargin(), itemSpacing(), margin(), spacing()

Definition at line 335 of file qwt_plot_legenditem.cpp.

14.86.3.13 itemSpacing()

```
int QwtPlotLegendItem::itemSpacing ( ) const
```

Returns

Spacing inside of each item

See also

setItemSpacing(), itemMargin(), margin(), spacing()

Definition at line 363 of file qwt_plot_legenditem.cpp.

14.86.3.14 legendGeometries() `QList< QRect > QwtPlotLegendItem::legendGeometries ( const QwtPlotItem * plotItem ) const`

Returns

Geometries of the items of a plot item

Note

Usually a plot item has only one entry on the legend

Definition at line 895 of file qwt_plot_legenditem.cpp.

14.86.3.15 margin() `int QwtPlotLegendItem::margin ( ) const`

Returns

Margin around the legend items

See also

setMargin(), spacing(), itemMargin(), itemSpacing()

Definition at line 280 of file qwt_plot_legenditem.cpp.

14.86.3.16 maxColumns() `uint QwtPlotLegendItem::maxColumns ( ) const`

Returns

Maximum number of columns

See also

maxColumns(), QwtDynGridLayout::maxColumns()

Definition at line 251 of file qwt_plot_legenditem.cpp.

14.86.3.17 minimumSize() `QSize QwtPlotLegendItem::minimumSize ( const QwtLegendData & data ) const [virtual]`

Minimum size hint needed to display an entry

Parameters

data	Attributes of the legend entry

Returns

Minimum size

Definition at line 823 of file qwt_plot_legenditem.cpp.

14.86.3.18 offsetInCanvas() `int QwtPlotLegendItem::offsetInCanvas ( Qt::Orientation orientation ) const`

Parameters

orientation	Qt::Horizontal is for the left/right, Qt::Vertical for the top/bottom padding.

Returns

Distance between the legend and the canvas border

See also

setOffsetInCanvas()

Definition at line 444 of file qwt_plot_legenditem.cpp.

14.86.3.19 plotItems() `QList< const QwtPlotItem * > QwtPlotLegendItem::plotItems ( ) const`

Returns

All plot items with an entry on the legend

Note

A plot item might have more than one entry on the legend

Definition at line 886 of file qwt_plot_legenditem.cpp.

14.86.3.20 rtti() `int QwtPlotLegendItem::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotLegend

Reimplemented from QwtPlotItem.

Definition at line 192 of file qwt_plot_legenditem.cpp.

14.86.3.21 setAlignmentInCanvas() `void QwtPlotLegendItem::setAlignmentInCanvas ( Qt::Alignment alignment )`

Set the alignmnet.

Alignment means the position of the legend relative to the geometry of the plot canvas.

Parameters

alignment	Alignment flags

See also

alignmentInCanvas(), setMaxColumns()

Note

To align a legend with many items horizontally the number of columns need to be limited

Definition at line 210 of file qwt_plot_legenditem.cpp.

14.86.3.22 setBackgroundBrush() `void QwtPlotLegendItem::setBackgroundBrush ( const QBrush & brush )`

Set the background brush.

The brush is used to fill the background

Parameters

brush	Brush

See also

backgroundBrush(), setBackgroundMode(), drawBackground()

Definition at line 509 of file qwt_plot_legenditem.cpp.

14.86.3.23 setBackgroundMode() `void QwtPlotLegendItem::setBackgroundMode ( BackgroundMode mode )`

Set the background mode.

Depending on the mode the complete legend or each item might have an background.

The default setting is LegendBackground.

See also

backgroundMode(), setBackgroundBrush(), drawBackground()

Definition at line 537 of file qwt_plot_legenditem.cpp.

14.86.3.24 setBorderPen() `void QwtPlotLegendItem::setBorderPen ( const QPen & pen )`

Set the pen for drawing the border

Parameters

pen	Border pen

See also

borderPen(), setBackgroundBrush()

Definition at line 483 of file qwt_plot_legenditem.cpp.

14.86.3.25 setBorderRadius() `void QwtPlotLegendItem::setBorderRadius ( double radius )`

Set the radius for the border

Parameters

radius	A value <= 0 defines a rectangular border

See also

borderRadius(), setBorderPen()

Definition at line 457 of file qwt_plot_legenditem.cpp.

14.86.3.26 setFont() `void QwtPlotLegendItem::setFont ( const QFont & font )`

Change the font used for drawing the text label

Parameters

font	Legend font

See also

font()

Definition at line 374 of file qwt_plot_legenditem.cpp.

14.86.3.27 setItemMargin() `void QwtPlotLegendItem::setItemMargin ( int margin )`

Set the margin around each item

Parameters

margin	Margin

See also

itemMargin(), setItemSpacing(), setMargin(), setSpacing()

Definition at line 319 of file qwt_plot_legenditem.cpp.

14.86.3.28 setItemSpacing() `void QwtPlotLegendItem::setItemSpacing ( int spacing )`

Set the spacing inside of each item

Parameters

spacing	Spacing

See also

itemSpacing(), setItemMargin(), setMargin(), setSpacing()

Definition at line 346 of file qwt_plot_legenditem.cpp.

14.86.3.29 setMargin() `void QwtPlotLegendItem::setMargin ( int margin )`

Set the margin around legend items.

The default setting for the margin is 0.

Parameters

margin	Margin in pixels

See also

margin(), setSpacing(), setItemMargin(), setItemSpacing

Definition at line 264 of file qwt_plot_legenditem.cpp.

14.86.3.30 setMaxColumns() `void QwtPlotLegendItem::setMaxColumns ( uint maxColumns )`

Limit the number of columns.

When aligning the legend horizontally (Qt::AlignLeft, Qt::AlignRight) the number of columns needs to be limited to avoid, that the width of the legend grows with an increasing number of entries.

Parameters

maxColumns	Maximum number of columns. 0 means unlimited.

See also

maxColumns(), QwtDynGridLayout::setMaxColumns()

Definition at line 238 of file qwt_plot_legenditem.cpp.

14.86.3.31 setOffsetInCanvas() `void QwtPlotLegendItem::setOffsetInCanvas ( Qt::Orientations orientations, int numPixels )`

Set the distance between the legend and the canvas border.

The default setting is 10 pixels.

Parameters

orientations	Qt::Horizontal is for the left/right, Qt::Vertical for the top/bottom offset.
numPixels	Distance in pixels

See also

setMargin()

Definition at line 405 of file qwt_plot_legenditem.cpp.

14.86.3.32 setSpacing() `void QwtPlotLegendItem::setSpacing ( int spacing )`

Set the spacing between the legend items.

Parameters

spacing	Spacing in pixels

See also

spacing(), setMargin()

Definition at line 294 of file qwt_plot_legenditem.cpp.

14.86.3.33 setTextPen() `void QwtPlotLegendItem::setTextPen ( const QPen & pen )`

Set the pen for drawing text labels.

Parameters

pen	Text pen

See also

textPen(), setFont()

Definition at line 561 of file qwt_plot_legenditem.cpp.

14.86.3.34 spacing() `int QwtPlotLegendItem::spacing ( ) const`

Returns

Spacing between the legend items

See also

setSpacing(), margin(), itemSpacing(), itemMargin()

Definition at line 308 of file qwt_plot_legenditem.cpp.

14.86.3.35 textPen() `QPen QwtPlotLegendItem::textPen ( ) const`

Returns

Pen for drawing text labels

See also

setTextPen(), font()

Definition at line 574 of file qwt_plot_legenditem.cpp.

14.86.3.36 updateLegend()

```
void QwtPlotLegendItem::updateLegend (
    const QwtPlotItem * plotItem,
    const QList< QwtLegendData > & data ) [override], [virtual]
```

Update the legend items according to modifications of a plot item

Parameters

plotItem	Plot item
data	Attributes of the legend entries

Reimplemented from QwtPlotItem.

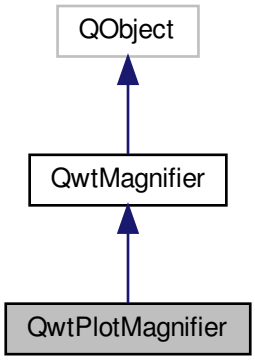
Definition at line 697 of file qwt_plot_legenditem.cpp.

14.87 QwtPlotMagnifier Class Reference

QwtPlotMagnifier provides zooming, by magnifying in steps.

`#include <qwt_plot_magnifier.h>`

Inheritance diagram for QwtPlotMagnifier:



Public Slots

- virtual void rescale (double factor) override

Public Member Functions

- QwtPlotMagnifier (QWidget ∗)
- virtual ∼QwtPlotMagnifier ()
  Destructor.
- void setAxisEnabled (QwtAxisId, bool on)
  En/Disable an axis.
- bool isAxisEnabled (QwtAxisId) const
- QWidget ∗ canvas ()
  Return observed plot canvas.
- const QWidget ∗ canvas () const
  Return Observed plot canvas.
- QwtPlot ∗ plot ()
  Return plot widget, containing the observed plot canvas.
- const QwtPlot ∗ plot () const
  Return plot widget, containing the observed plot canvas.

Additional Inherited Members

14.87.1 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

Definition at line 30 of file qwt_plot_magnifier.h.

14.87.2 Constructor & Destructor Documentation

14.87.2.1 QwtPlotMagnifier() `QwtPlotMagnifier::QwtPlotMagnifier ( QWidget * canvas ) [explicit]`

Constructor

Parameters

canvas	Plot canvas to be magnified

Definition at line 30 of file qwt_plot_magnifier.cpp.

14.87.3 Member Function Documentation

14.87.3.1 isAxisEnabled() `bool QwtPlotMagnifier::isAxisEnabled ( QwtAxisId axisId ) const`

Test if an axis is enabled

Parameters

axis↩ Id	Axis

Returns

True, if the axis is enabled

See also

setAxisEnabled()

Definition at line 67 of file qwt_plot_magnifier.cpp.

14.87.3.2 rescale `void QwtPlotMagnifier::rescale ( double factor ) [override], [virtual], [slot]`

Zoom in/out the axes scales

Parameters

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

Definition at line 111 of file qwt_plot_magnifier.cpp.

14.87.3.3 setAxisEnabled() `void QwtPlotMagnifier::setAxisEnabled ( QwtAxisId axisId, bool on )`

En/Disable an axis.

Only Axes that are enabled will be zoomed. All other axes will remain unchanged.

Parameters

axisId	Axis
on	On/Off

See also

isAxisEnabled()

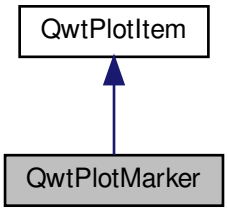
Definition at line 53 of file qwt_plot_magnifier.cpp.

14.88 QwtPlotMarker Class Reference

A class for drawing markers.

`#include <qwt_plot_marker.h>`

Inheritance diagram for QwtPlotMarker:



Public Types

- enum LineStyle { NoLine , HLine , VLine , Cross }

Public Member Functions

- QwtPlotMarker ()

  Sets alignment to Qt::AlignCenter, and style to QwtPlotMarker::NoLine.
- QwtPlotMarker (const QString &title)

  Sets alignment to Qt::AlignCenter, and style to QwtPlotMarker::NoLine.
- QwtPlotMarker (const QwtText &title)

  Sets alignment to Qt::AlignCenter, and style to QwtPlotMarker::NoLine.
- virtual ~QwtPlotMarker ()

  Destructor.
- virtual int rtti () const override
- double xValue () const

  Return x Value.
- double yValue () const

  Return y Value.
- QPointF value () const

  Return Value.
- void setXValue (double)

  Set X Value.
- void setYValue (double)

  Set Y Value.
- void setValue (double, double)

  Set Value.
- void setValue (const QPointF &)

  Set Value.
- void setLineStyle (LineStyle)

  Set the line style.
- LineStyle lineStyle () const
- void setLinePen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setLinePen (const QPen &)
- const QPen & linePen () const
- void setSymbol (const QwtSymbol *)

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

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

  Set the alignment of the label.
- Qt::Alignment labelAlignment () const
- void setLabelOrientation (Qt::Orientation)

  Set the orientation of the label.
- Qt::Orientation labelOrientation () const
- void setSpacing (int)

  Set the spacing.
- int spacing () const
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &) const override
- virtual QRectF boundingRect () const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override

Protected Member Functions

- virtual void drawLines (QPainter *, const QRectF &, const QPointF &) const
- virtual void drawSymbol (QPainter *, const QRectF &, const QPointF &) const
- virtual void drawLabel (QPainter *, const QRectF &, const QPointF &) const

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

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

Note

QwtPlotTextLabel is intended to align a text label according to the geometry of canvas (unrelated to plot coordinates)

Definition at line 45 of file qwt_plot_marker.h.

14.88.2 Member Enumeration Documentation

14.88.2.1 LineStyle `enum QwtPlotMarker::LineStyle`

Line styles.

See also

setLineStyle(), lineStyle()

Enumerator

NoLine	No line.
HLine	A horizontal line.
VLine	A vertical line.
Cross	A crosshair.

Definition at line 53 of file qwt_plot_marker.h.

14.88.3 Member Function Documentation

14.88.3.1 boundingRect() `QRectF QwtPlotMarker::boundingRect ( ) const [override], [virtual]`

Returns

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

Note

A width or height < 0.0 is ignored by the autoscaler

Reimplemented from QwtPlotItem.

Definition at line 572 of file qwt_plot_marker.cpp.

14.88.3.2 draw()

```
void QwtPlotMarker::draw (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const [override], [virtual]
```

Draw the marker

Parameters

painter	Painter
xMap	x Scale Map
yMap	y Scale Map
canvasRect	Contents rectangle of the canvas in painter coordinates

Implements QwtPlotItem.

Definition at line 142 of file qwt_plot_marker.cpp.

14.88.3.3 drawLabel()

```
void QwtPlotMarker::drawLabel (
    QPainter * painter,
    const QRectF & canvasRect,
    const QPointF & pos ) const [protected], [virtual]
```

Align and draw the text label of the marker

Parameters

painter	Painter
canvasRect	Contents rectangle of the canvas in painter coordinates
pos	Position of the marker, translated into widget coordinates

See also

drawLabel(), drawSymbol()

Definition at line 232 of file qwt_plot_marker.cpp.

14.88.3.4 drawLines()

```
void QwtPlotMarker::drawLines (
    QPainter * painter,
    const QRectF & canvasRect,
    const QPointF & pos ) const  [protected], [virtual]
```

Draw the lines marker

Parameters

painter	Painter
canvasRect	Contents rectangle of the canvas in painter coordinates
pos	Position of the marker, translated into widget coordinates

See also

drawLabel(), drawSymbol()

Definition at line 163 of file qwt_plot_marker.cpp.

14.88.3.5 drawSymbol()

```
void QwtPlotMarker::drawSymbol (
    QPainter * painter,
    const QRectF & canvasRect,
    const QPointF & pos ) const  [protected], [virtual]
```

Draw the symbol of the marker

Parameters

painter	Painter
canvasRect	Contents rectangle of the canvas in painter coordinates
pos	Position of the marker, translated into widget coordinates

See also

drawLabel(), QwtSymbol::drawSymbol()

Definition at line 203 of file qwt_plot_marker.cpp.

14.88.3.6 label() `QwtText QwtPlotMarker::label ( ) const`

Returns

the label

See also

setLabel()

Definition at line 433 of file qwt_plot_marker.cpp.

14.88.3.7 labelAlignment() `Qt::Alignment QwtPlotMarker::labelAlignment ( ) const`

Returns

the label alignment

See also

setLabelAlignment(), setLabelOrientation()

Definition at line 465 of file qwt_plot_marker.cpp.

14.88.3.8 labelOrientation() `Qt::Orientation QwtPlotMarker::labelOrientation ( ) const`

Returns

the label orientation

See also

setLabelOrientation(), labelAlignment()

Definition at line 493 of file qwt_plot_marker.cpp.

14.88.3.9 legendIcon()

```
QwtGraphic QwtPlotMarker::legendIcon (
        int index,
        const QSizeF & size ) const  [override], [virtual]
```

Returns

Icon representing the marker on the legend

Parameters

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

See also

setLegendIconSize(), legendData()

Reimplemented from QwtPlotItem.

Definition at line 598 of file qwt_plot_marker.cpp.

14.88.3.10 linePen() `const QPen & QwtPlotMarker::linePen ( ) const`

Returns

the line pen

See also

setLinePen()

Definition at line 567 of file qwt_plot_marker.cpp.

14.88.3.11 lineStyle() `QwtPlotMarker::LineStyle QwtPlotMarker::lineStyle ( ) const`

Returns

the line style

See also

setLineStyle()

Definition at line 381 of file qwt_plot_marker.cpp.

14.88.3.12 rtti() `int QwtPlotMarker::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotMarker

Reimplemented from QwtPlotItem.

Definition at line 82 of file qwt_plot_marker.cpp.

14.88.3.13 setLabel() `void QwtPlotMarker::setLabel ( const QwtText & label )`

Set the label.

Parameters

label	Label text

See also

label()

Definition at line 420 of file qwt_plot_marker.cpp.

14.88.3.14 setLabelAlignment() `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

align	Alignment.

See also

labelAlignment(), labelOrientation()

Definition at line 452 of file qwt_plot_marker.cpp.

14.88.3.15 setLabelOrientation() `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

orientation	Orientation of the label

See also

labelOrientation(), setLabelAlignment()

Definition at line 480 of file qwt_plot_marker.cpp.

14.88.3.16 setLinePen() [1/2] void QwtPlotMarker::setLinePen (const QColor & *color*, qreal *width* = *0.0*, Qt::PenStyle *style* = *Qt::SolidLine*)

Build and assign a line pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 541 of file qwt_plot_marker.cpp.

14.88.3.17 setLinePen() [2/2] void QwtPlotMarker::setLinePen (const QPen & *pen*)

Specify a pen for the line.

Parameters

pen	New pen

See also

linePen()

Definition at line 552 of file qwt_plot_marker.cpp.

14.88.3.18 setLineStyle() void QwtPlotMarker::setLineStyle (LineStyle *style*)

Set the line style.

Parameters

style	Line style.

See also

lineStyle()

Definition at line 366 of file qwt_plot_marker.cpp.

14.88.3.19 setSpacing() `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

spacing	Spacing

See also

spacing(), setLabelAlignment()

Definition at line 507 of file qwt_plot_marker.cpp.

14.88.3.20 setSymbol() `void QwtPlotMarker::setSymbol ( const QwtSymbol * symbol )`

Assign a symbol.

Parameters

symbol	New symbol

See also

symbol()

Definition at line 391 of file qwt_plot_marker.cpp.

14.88.3.21 spacing() `int QwtPlotMarker::spacing ( ) const`

Returns

the spacing

See also

setSpacing()

Definition at line 523 of file qwt_plot_marker.cpp.

14.88.3.22 symbol() `const QwtSymbol * QwtPlotMarker::symbol ( ) const`

Returns

the symbol

See also

setSymbol(), QwtSymbol

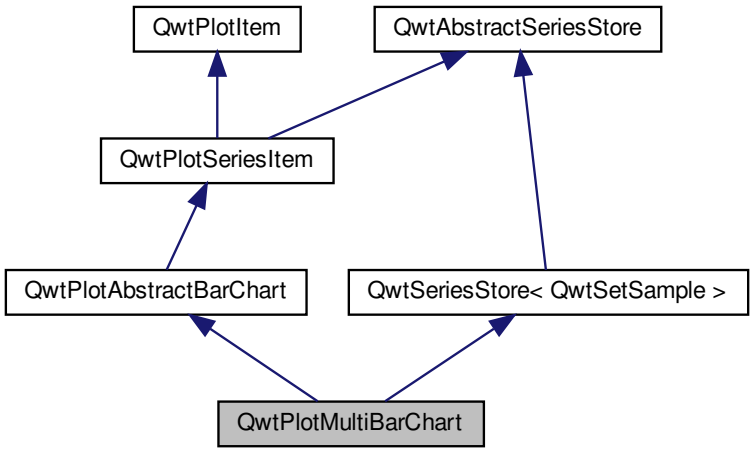
Definition at line 410 of file qwt_plot_marker.cpp.

14.89 QwtPlotMultiBarChart Class Reference

QwtPlotMultiBarChart displays a series of a samples that consist each of a set of values.

`#include <qwt_plot_multi_barchart.h>`

Inheritance diagram for QwtPlotMultiBarChart:



Public Types

- enum ChartStyle { Grouped , Stacked }

  Chart styles.

Public Member Functions

- QwtPlotMultiBarChart (const QString &title=QString())
- QwtPlotMultiBarChart (const QwtText &title)
- virtual ~QwtPlotMultiBarChart ()

  Destructor.
- virtual int rtti () const override
- void setBarTitles (const QList< QwtText > &)

  Set the titles for the bars.
- QList< QwtText > barTitles () const
- void setSamples (const QVector< QwtSetSample > &)
- void setSamples (const QVector< QVector< double > > &)
- void setSamples (QwtSeriesData< QwtSetSample > *)
- void setStyle (ChartStyle style)
- ChartStyle style () const
- void setSymbol (int valueIndex, QwtColumnSymbol *)

  Add a symbol to the symbol map.
- const QwtColumnSymbol * symbol (int valueIndex) const
- void resetSymbolMap ()
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual QRectF boundingRect () const override
- virtual QList< QwtLegendData > legendData () const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override

Protected Member Functions

- QwtColumnSymbol * symbol (int valueIndex)
- virtual QwtColumnSymbol * specialSymbol (int sampleIndex, int valueIndex) const

  Create a symbol for special values.
- virtual void drawSample (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, const QwtInterval &boundingInterval, int index, const QwtSetSample &) const
- virtual void drawBar (QPainter *, int sampleIndex, int valueIndex, const QwtColumnRect &) const
- void drawStackedBars (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int index, double sampleWidth, const QwtSetSample &) const
- void drawGroupedBars (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int index, double sampleWidth, const QwtSetSample &) const

14.89.1 Detailed Description

QwtPlotMultiBarChart displays a series of a samples that consist each of a set of values.

Each value is displayed as a bar, the bars of each set can be organized side by side or accumulated.

Each bar of a set is rendered by a QwtColumnSymbol, that is set by setSymbol(). The bars of different sets use the same symbols. Exceptions are possible by overloading specialSymbol() or overloading drawBar().

Depending on its orientation() the bars are displayed horizontally or vertically. The bars cover the interval between the baseline() and the value.

In opposite to most other plot items, QwtPlotMultiBarChart returns more than one entry for the legend - one for each symbol.

See also

QwtPlotBarChart, QwtPlotHistogram QwtPlotSeriesItem::orientation(), QwtPlotAbstractBarChart::baseline()

Definition at line 41 of file qwt_plot_multi_barchart.h.

14.89.2 Member Enumeration Documentation

14.89.2.1 ChartStyle `enum QwtPlotMultiBarChart::ChartStyle`

Chart styles.

The default setting is QwtPlotMultiBarChart::Grouped.

See also

setStyle(), style()

Enumerator

Grouped	The bars of a set are displayed side by side.
Stacked	The bars are displayed on top of each other accumulating to a single bar. All values of a set need to have the same sign.

Definition at line 52 of file qwt_plot_multi_barchart.h.

14.89.3 Constructor & Destructor Documentation

14.89.3.1 QwtPlotMultiBarChart() [1/2]

```
QwtPlotMultiBarChart::QwtPlotMultiBarChart (
    const QString & title = QString() ) [explicit]
```

Constructor

Parameters

title	Title of the chart

Definition at line 62 of file qwt_plot_multi_barchart.cpp.

14.89.3.2 QwtPlotMultiBarChart() [2/2]

```
QwtPlotMultiBarChart::QwtPlotMultiBarChart (
        const QwtText & title ) [explicit]
```

Constructor

Parameters

title	Title of the chart

Definition at line 52 of file qwt_plot_multi_barchart.cpp.

14.89.4 Member Function Documentation

14.89.4.1 barTitles()

```
QList< QwtText > QwtPlotMultiBarChart::barTitles ( ) const
```

Returns

Bar titles

See also

setBarTitles(), legendData()

Definition at line 148 of file qwt_plot_multi_barchart.cpp.

14.89.4.2 boundingRect()

```
QRectF QwtPlotMultiBarChart::boundingRect ( ) const [override], [virtual]
```

Returns

Bounding rectangle of all samples. For an empty series the rectangle is invalid.

Reimplemented from QwtPlotSeriesItem.

Definition at line 302 of file qwt_plot_multi_barchart.cpp.

14.89.4.3 drawBar()

```
void QwtPlotMultiBarChart::drawBar (
        QPainter * painter,
        int sampleIndex,
        int valueIndex,
        const QwtColumnRect & rect ) const [protected], [virtual]
```

Draw a bar

Parameters

painter	Painter
sampleIndex	Index of the sample - might be -1 when the bar is painted for the legend
valueIndex	Index of a value in a set
rect	Directed target rectangle for the bar

See also

drawSeries()

Definition at line 652 of file qwt_plot_multi_barchart.cpp.

14.89.4.4 drawGroupedBars()

```
void QwtPlotMultiBarChart::drawGroupedBars (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int index,
    double sampleWidth,
    const QwtSetSample & sample ) const  [protected]
```

Draw a grouped sample

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
index	Index of the sample to be painted
sampleWidth	Bounding width for all bars of the sample
sample	Sample

See also

drawSeries(), sampleWidth()

Definition at line 461 of file qwt_plot_multi_barchart.cpp.

14.89.4.5 drawSample()

```
void QwtPlotMultiBarChart::drawSample (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    const QwtInterval & boundingInterval,
```

```
int index,
const QwtSetSample & sample ) const  [protected], [virtual]
```

Draw a sample

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
boundingInterval	Bounding interval of sample values
index	Index of the sample to be painted
sample	Sample value

See also

drawSeries()

Definition at line 415 of file qwt_plot_multi_barchart.cpp.

14.89.4.6 drawSeries()

```
void QwtPlotMultiBarChart::drawSeries (
        QPainter * painter,
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect,
        int from,
        int to ) const  [override], [virtual]
```

Draw an interval of the bar chart

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

See also

drawSymbols()

Implements QwtPlotSeriesItem.

Definition at line 374 of file qwt_plot_multi_barchart.cpp.

14.89.4.7 drawStackedBars()

```
void QwtPlotMultiBarChart::drawStackedBars (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int index,
    double sampleWidth,
    const QwtSetSample & sample ) const  [protected]
```

Draw a stacked sample

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
index	Index of the sample to be painted
sampleWidth	Width of the bars
sample	Sample

See also

drawSeries(), sampleWidth()

Definition at line 541 of file qwt_plot_multi_barchart.cpp.

14.89.4.8 legendData()

```
QList< QwtLegendData > QwtPlotMultiBarChart::legendData ( ) const  [override], [virtual]
```

Returns

Information to be displayed on the legend

The chart is represented by a list of entries - one for each bar title. Each element contains a bar title and an icon showing its corresponding bar.

See also

barTitles(), legendIcon(), legendIconSize()

Reimplemented from QwtPlotItem.

Definition at line 687 of file qwt_plot_multi_barchart.cpp.

14.89.4.9 legendIcon()

```
QwtGraphic QwtPlotMultiBarChart::legendIcon (
    int index,
    const QSizeF & size ) const  [override], [virtual]
```

Returns

Icon for representing a bar on the legend

Parameters

index	Index of the bar
size	Icon size

Returns

An icon showing a bar

See also

drawBar(), legendData()

Reimplemented from QwtPlotItem.

Definition at line 720 of file qwt_plot_multi_barchart.cpp.

14.89.4.10 resetSymbolMap() `void QwtPlotMultiBarChart::resetSymbolMap ( )`

Remove all symbols from the symbol map

Definition at line 237 of file qwt_plot_multi_barchart.cpp.

14.89.4.11 rtti() `int QwtPlotMultiBarChart::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotBarChart

Reimplemented from QwtPlotItem.

Definition at line 82 of file qwt_plot_multi_barchart.cpp.

14.89.4.12 setBarTitles() `void QwtPlotMultiBarChart::setBarTitles ( const QList< QwtText > & titles )`

Set the titles for the bars.

The titles are used for the legend.

Parameters

titles	Bar titles

See also

barTitles(), legendData()

Definition at line 138 of file qwt_plot_multi_barchart.cpp.

14.89.4.13 setSamples() [1/3] void QwtPlotMultiBarChart::setSamples (const QVector< QVector< double > > & *samples*)

Initialize data with an array of samples.

Parameters

samples	Vector of points

Definition at line 101 of file qwt_plot_multi_barchart.cpp.

14.89.4.14 setSamples() [2/3] void QwtPlotMultiBarChart::setSamples (const QVector< QwtSetSample > & *samples*)

Initialize data with an array of samples.

Parameters

samples	Vector of points

Definition at line 91 of file qwt_plot_multi_barchart.cpp.

14.89.4.15 setSamples() [3/3] void QwtPlotMultiBarChart::setSamples (QwtSeriesData< QwtSetSample > * *data*)

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 123 of file qwt_plot_multi_barchart.cpp.

14.89.4.16 setStyle() void QwtPlotMultiBarChart::setStyle (ChartStyle *style*)

Set the style of the chart

Parameters

style	Chart style

See also

style()

Definition at line 278 of file qwt_plot_multi_barchart.cpp.

14.89.4.17 setSymbol() void QwtPlotMultiBarChart::setSymbol (int *valueIndex*, QwtColumnSymbol * *symbol*)

Add a symbol to the symbol map.

Assign a default symbol for drawing the bar representing all values with the same index in a set.

Parameters

valueIndex	Index of a value in a set
symbol	Symbol used for drawing a bar

See also

symbol(), resetSymbolMap(), specialSymbol()

Definition at line 164 of file qwt_plot_multi_barchart.cpp.

14.89.4.18 specialSymbol() QwtColumnSymbol * QwtPlotMultiBarChart::specialSymbol (int *sampleIndex*, int *valueIndex*) const [protected], [virtual]

Create a symbol for special values.

Usually the symbols for displaying a bar are set by setSymbols() and common for all sets. By overloading specialSymbol() it is possible to create a temporary symbol() for displaying a special value.

The symbol has to be created by new each time specialSymbol() is called. As soon as the symbol is painted this symbol gets deleted.

When no symbol (NULL) is returned, the value will be displayed with the standard symbol that is used for all symbols with the same valueIndex.

Parameters

sampleIndex	Index of the sample
valueIndex	Index of the value in the set

Returns

NULL, meaning that the value is not special

Definition at line 263 of file qwt_plot_multi_barchart.cpp.

14.89.4.19 style() `QwtPlotMultiBarChart::ChartStyle QwtPlotMultiBarChart::style ( ) const`

Returns

Style of the chart

See also

setStyle()

Definition at line 293 of file qwt_plot_multi_barchart.cpp.

14.89.4.20 symbol() [1/2] `QwtColumnSymbol * QwtPlotMultiBarChart::symbol ( int valueIndex ) [protected]`

Find a symbol in the symbol map

Parameters

valueIndex	Index of a value in a set

Returns

The symbol, that had been set by setSymbol() or NULL.

See also

setSymbol(), specialSymbol(), drawBar()

Definition at line 226 of file qwt_plot_multi_barchart.cpp.

14.89.4.21 symbol() [2/2] `const QwtColumnSymbol * QwtPlotMultiBarChart::symbol ( int valueIndex ) const`

Find a symbol in the symbol map

Parameters

valueIndex	Index of a value in a set

Returns

The symbol, that had been set by setSymbol() or NULL.

See also

setSymbol(), specialSymbol(), drawBar()

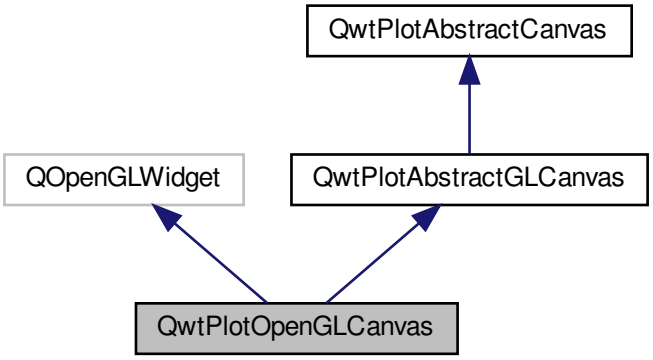
Definition at line 210 of file qwt_plot_multi_barchart.cpp.

14.90 QwtPlotOpenGLCanvas Class Reference

An alternative canvas for a QwtPlot derived from QOpenGLWidget.

`#include <qwt_plot_opengl_canvas.h>`

Inheritance diagram for QwtPlotOpenGLCanvas:



Public Slots

- void replot ()

Public Member Functions

- QwtPlotOpenGLCanvas (QwtPlot *=NULL)
  Constructor.
- QwtPlotOpenGLCanvas (const QSurfaceFormat &, QwtPlot *=NULL)
  Constructor.
- virtual ~QwtPlotOpenGLCanvas ()
  Destructor.
- virtual Q_INVOKABLE void invalidateBackingStore () override
  Invalidate the internal backing store.
- Q_INVOKABLE QPainterPath borderPath (const QRect &) const
- virtual bool event (QEvent *) override

Protected Member Functions

- virtual void paintEvent (QPaintEvent *) override
- virtual void initializeGL () override
  No operation - reserved for some potential use in the future.
- virtual void paintGL () override
  Paint the plot.
- virtual void resizeGL (int width, int height) override
  No operation - reserved for some potential use in the future.

Additional Inherited Members

14.90.1 Detailed Description

An alternative canvas for a QwtPlot derived from QOpenGLWidget.

Even if QwtPlotOpenGLCanvas is not derived from QFrame it imitates its API. When using style sheets it supports the box model - beside backgrounds with rounded borders.

See also

QwtPlot::setCanvas(), QwtPlotCanvas, QwtPlotCanvas::OpenGLBuffer

Note

Another way for getting hardware accelerated graphics is using an OpenGL offscreen buffer (QwtPlot↩ Canvas::OpenGLBuffer) with QwtPlotCanvas. Performance is worse, than rendering straight to a QOpen↩ GLWidget, but is usually better integrated into a desktop application.

Definition at line 34 of file qwt_plot_opengl_canvas.h.

14.90.2 Constructor & Destructor Documentation

14.90.2.1 QwtPlotOpenGLCanvas() [1/2] QwtPlotOpenGLCanvas::QwtPlotOpenGLCanvas (QwtPlot * *plot* = *NULL*) [explicit]

Constructor.

Parameters

plot	Parent plot widget

See also

QwtPlot::setCanvas()

Definition at line 48 of file qwt_plot_opengl_canvas.cpp.

14.90.2.2 QwtPlotOpenGLCanvas() [2/2]

```
QwtPlotOpenGLCanvas::QwtPlotOpenGLCanvas (
    const QSurfaceFormat & format,
    QwtPlot * plot = NULL )  [explicit]
```

Constructor.

Parameters

format	OpenGL surface format
plot	Parent plot widget

See also

QwtPlot::setCanvas()

Definition at line 65 of file qwt_plot_opengl_canvas.cpp.

14.90.3 Member Function Documentation

14.90.3.1 borderPath()

```
QPainterPath QwtPlotOpenGLCanvas::borderPath (
    const QRect & rect ) const
```

Calculate the painter path for a styled or rounded border

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

Parameters

rect	Bounding rectangle of the canvas

Returns

Painter path, that can be used for clipping

Definition at line 168 of file qwt_plot_opengl_canvas.cpp.

14.90.3.2 event() `bool QwtPlotOpenGLCanvas::event ( QEvent * event ) [override], [virtual]`

Qt event handler for QEvent::PolishRequest and QEvent::StyleChange

Parameters

event	Qt Event

Returns

See QGLWidget::event()

Definition at line 112 of file qwt_plot_opengl_canvas.cpp.

14.90.3.3 paintEvent() `void QwtPlotOpenGLCanvas::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Paint event

Parameters

event	Paint event

See also

QwtPlot::drawCanvas()

Definition at line 101 of file qwt_plot_opengl_canvas.cpp.

14.90.3.4 replot `void QwtPlotOpenGLCanvas::replot ( ) [slot]`

Invalidate the paint cache and repaint the canvas

See also

invalidatePaintCache()

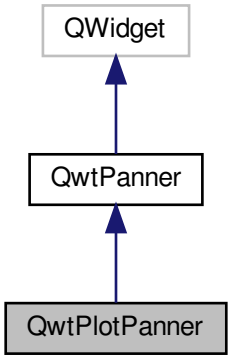
Definition at line 142 of file qwt_plot_opengl_canvas.cpp.

14.91 QwtPlotPanner Class Reference

QwtPlotPanner provides panning of a plot canvas.

`#include <qwt_plot_panner.h>`

Inheritance diagram for QwtPlotPanner:



Public Slots

- virtual void moveCanvas (int dx, int dy)

Public Member Functions

- QwtPlotPanner (QWidget ∗)
  A panner for the canvas of a QwtPlot.
- virtual ∼QwtPlotPanner ()
  Destructor.
- QWidget ∗ canvas ()
  Return observed plot canvas.
- const QWidget ∗ canvas () const
  Return Observed plot canvas.
- QwtPlot ∗ plot ()
  Return plot widget, containing the observed plot canvas.
- const QwtPlot ∗ plot () const
  Return plot widget, containing the observed plot canvas.
- void setAxisEnabled (QwtAxisId axisId, bool on)
  En/Disable an axis.
- bool isAxisEnabled (QwtAxisId) const

Protected Member Functions

- virtual QBitmap contentsMask () const override
- virtual QPixmap grab () const override

Additional Inherited Members

14.91.1 Detailed Description

QwtPlotPanner provides panning of a plot canvas.

QwtPlotPanner is a panner for a plot canvas, 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

Definition at line 32 of file qwt_plot_panner.h.

14.91.2 Constructor & Destructor Documentation

14.91.2.1 QwtPlotPanner() `QwtPlotPanner::QwtPlotPanner ( QWidget * canvas ) [explicit]`

A panner for the canvas of a QwtPlot.

The panner is enabled for all axes

Parameters

canvas	Plot canvas to pan, also the parent object

See also

setAxisEnabled()

Definition at line 128 of file qwt_plot_panner.cpp.

14.91.3 Member Function Documentation

14.91.3.1 contentsMask() `QBitmap QwtPlotPanner::contentsMask ( ) const [override], [protected], [virtual]`

Calculate a mask from the border path of the canvas

Returns

Mask as bitmap

See also

QwtPlotCanvas::borderPath()

Reimplemented from QwtPanner.

Definition at line 267 of file qwt_plot_panner.cpp.

14.91.3.2 grab() `QPixmap QwtPlotPanner::grab ( ) const [override], [protected], [virtual]`

Returns

Pixmap with the content of the canvas

Reimplemented from QwtPanner.

Definition at line 278 of file qwt_plot_panner.cpp.

14.91.3.3 isAxisEnabled() `bool QwtPlotPanner::isAxisEnabled ( QwtAxisId axisId ) const`

Test if an axis is enabled

Parameters

axisId	Axis

Returns

True, if the axis is enabled

See also

setAxisEnabled(), moveCanvas()

Definition at line 168 of file qwt_plot_panner.cpp.

14.91.3.4 moveCanvas

```
void QwtPlotPanner::moveCanvas (
    int dx,
    int dy ) [virtual], [slot]
```

Adjust the enabled axes according to dx/dy

Parameters

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

See also

QwtPanner::panned()

Definition at line 216 of file qwt_plot_panner.cpp.

14.91.3.5 setAxisEnabled()

```
void QwtPlotPanner::setAxisEnabled (
    QwtAxisId axisId,
    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

axis↩ Id	Axis id
on	On/Off

See also

isAxisEnabled(), moveCanvas()

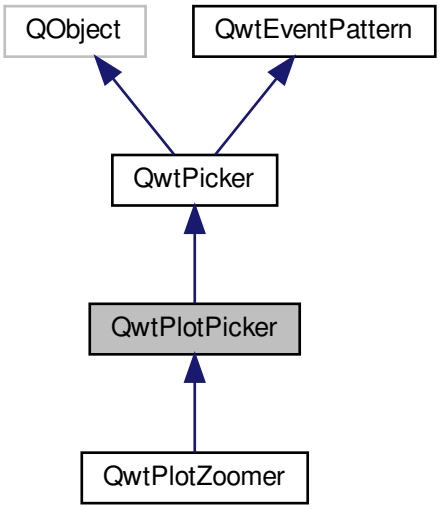
Definition at line 154 of file qwt_plot_panner.cpp.

14.92 QwtPlotPicker Class Reference

QwtPlotPicker provides selections on a plot canvas.

```
#include <qwt_plot_picker.h>
```

Inheritance diagram for QwtPlotPicker:



Signals

- void selected (const QPointF &pos)
- void selected (const QRectF &rect)
- void selected (const QVector< QPointF > &pa)
- void appended (const QPointF &pos)
- void moved (const QPointF &pos)

Public Member Functions

- QwtPlotPicker (QWidget *canvas)

  Create a plot picker.
- virtual ~QwtPlotPicker ()

  Destructor.
- QwtPlotPicker (QwtAxisId xAxisId, QwtAxisId yAxisId, QWidget *)
- QwtPlotPicker (QwtAxisId xAxisId, QwtAxisId yAxisId, RubberBand rubberBand, DisplayMode trackerMode, QWidget *)
- virtual void setAxes (QwtAxisId xAxisId, QwtAxisId yAxisId)
- QwtAxisId xAxis () const

  Return x axis.
- QwtAxisId yAxis () const

  Return y axis.
- QwtPlot * plot ()
- const QwtPlot * plot () const
- QWidget * canvas ()
- const QWidget * canvas () const

Protected Member Functions

- QRectF scaleRect () const
- QRectF invTransform (const QRect &) const
- QRect transform (const QRectF &) const
- QPointF invTransform (const QPoint &) const
- QPoint transform (const QPointF &) const
- virtual QwtText trackerText (const QPoint &) const override
- virtual QwtText trackerTextF (const QPointF &) const

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

Additional Inherited Members

14.92.1 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.

Definition at line 33 of file qwt_plot_picker.h.

14.92.2 Constructor & Destructor Documentation

14.92.2.1 QwtPlotPicker() [1/3] QwtPlotPicker::QwtPlotPicker (QWidget * *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 QwtAxis::XBottom. If both or no y-axis are enabled, it is set to QwtAxis::YLeft.

Parameters

canvas	Plot canvas to observe, also the parent object

See also

QwtPlot::autoReplot(), QwtPlot::replot(), scaleRect()

Definition at line 43 of file qwt_plot_picker.cpp.

14.92.2.2 QwtPlotPicker() [2/3]

```
QwtPlotPicker::QwtPlotPicker (
    QwtAxisId xAxisId,
    QwtAxisId yAxisId,
    QWidget * canvas )  [explicit]
```

Create a plot picker

Parameters

x↩ AxisId	X axis of the picker
y↩ AxisId	Y axis of the picker
canvas	Plot canvas to observe, also the parent object

See also

QwtPlot::autoReplot(), QwtPlot::replot(), scaleRect()

Definition at line 76 of file qwt_plot_picker.cpp.

14.92.2.3 QwtPlotPicker() [3/3]

```
QwtPlotPicker::QwtPlotPicker (
    QwtAxisId xAxisId,
    QwtAxisId yAxisId,
    RubberBand rubberBand,
    DisplayMode trackerMode,
    QWidget * canvas )  [explicit]
```

Create a plot picker

Parameters

xAxis	X axis of the picker
yAxis	Y axis of the picker
rubberBand	Rubber band style
trackerMode	Tracker mode
canvas	Plot canvas to observe, also the parent object

See also

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

Definition at line 98 of file qwt_plot_picker.cpp.

14.92.3 Member Function Documentation

14.92.3.1 append() `void QwtPlotPicker::append ( const QPoint & pos ) [override], [protected], [virtual]`

Append a point to the selection and update rubber band and tracker.

Parameters

pos	Additional point

See also

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

Note

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

Reimplemented from QwtPicker.

Definition at line 251 of file qwt_plot_picker.cpp.

14.92.3.2 appended `void QwtPlotPicker::appended ( const QPointF & pos ) [signal]`

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

Parameters

pos	Position of the appended point.

See also

append(). moved()

14.92.3.3 canvas() [1/2] `QWidget * QwtPlotPicker::canvas ( )`

Returns

Observed plot canvas

Definition at line 114 of file qwt_plot_picker.cpp.

14.92.3.4 canvas() [2/2] const QWidget * QwtPlotPicker::canvas ( ) const

Returns

Observed plot canvas

Definition at line 120 of file qwt_plot_picker.cpp.

14.92.3.5 end() bool QwtPlotPicker::end (bool *ok* = *true*) [override], [protected], [virtual]

Close a selection setting the state to inactive.

Parameters

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

Returns

True if the selection has been accepted, false otherwise

Reimplemented from QwtPicker.

Reimplemented in QwtPlotZoomer.

Definition at line 280 of file qwt_plot_picker.cpp.

14.92.3.6 invTransform() [1/2] QPointF QwtPlotPicker::invTransform (const QPoint & *pos*) const [protected]

Translate a point from pixel into plot coordinates

Returns

Point in plot coordinates

See also

transform()

Definition at line 367 of file qwt_plot_picker.cpp.

14.92.3.7 invTransform() [2/2] QRectF QwtPlotPicker::invTransform (const QRect & *rect*) const [protected]

Translate a rectangle from pixel into plot coordinates

Returns

Rectangle in plot coordinates

See also

transform()

Definition at line 341 of file qwt_plot_picker.cpp.

14.92.3.8 move() void QwtPlotPicker::move (const QPoint & *pos*) [override], [protected], [virtual]

Move the last point of the selection

Parameters

pos	New position

See also

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

Note

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

Reimplemented from QwtPicker.

Definition at line 266 of file qwt_plot_picker.cpp.

14.92.3.9 moved void QwtPlotPicker::moved (const QPointF & *pos*) [signal]

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

Parameters

pos	Position of the moved last point of the selection.

See also

move(), appended()

14.92.3.10 plot() [1/2] QwtPlot * QwtPlotPicker::plot ( )

Returns

Plot widget, containing the observed plot canvas

Definition at line 126 of file qwt_plot_picker.cpp.

14.92.3.11 plot() [2/2] const QwtPlot * QwtPlotPicker::plot ( ) const

Returns

Plot widget, containing the observed plot canvas

Definition at line 136 of file qwt_plot_picker.cpp.

14.92.3.12 scaleRect() QRectF QwtPlotPicker::scaleRect ( ) const [protected]

Returns

Normalized bounding rectangle of the axes

See also

QwtPlot::autoReplot(), QwtPlot::replot().

Definition at line 149 of file qwt_plot_picker.cpp.

14.92.3.13 selected [1/3] void QwtPlotPicker::selected (const QPointF & *pos*) [signal]

A signal emitted in case of QwtPickerMachine::PointSelection.

Parameters

pos	Selected point

14.92.3.14 selected [2/3] void QwtPlotPicker::selected (const QRectF & *rect*) [signal]

A signal emitted in case of QwtPickerMachine::RectSelection.

Parameters

rect	Selected rectangle

14.92.3.15 selected [3/3] void QwtPlotPicker::selected (const QVector< QPointF > & *pa*) [signal]

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

Parameters

pa	Selected points

14.92.3.16 setAxes() void QwtPlotPicker::setAxes (QwtAxisId *xAxisId*, QwtAxisId *yAxisId*) [virtual]

Set the x and y axes of the picker

Parameters

x↩ AxisId	X axis
y↩ AxisId	Y axis

Reimplemented in QwtPlotZoomer.

Definition at line 172 of file qwt_plot_picker.cpp.

14.92.3.17 trackerText() QwtText QwtPlotPicker::trackerText (const QPoint & *pos*) const [override], [protected], [virtual]

Translate a pixel position into a position string

Parameters

pos	Position in pixel coordinates

Returns

Position string

Reimplemented from QwtPicker.

Definition at line 203 of file qwt_plot_picker.cpp.

14.92.3.18 trackerTextF() `QwtText QwtPlotPicker::trackerTextF ( const QPointF & pos ) const [protected], [virtual]`

Translate a position into a position string.

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

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

Parameters

pos	Position

Returns

Position string

Definition at line 223 of file qwt_plot_picker.cpp.

14.92.3.19 transform() [1/2] `QPoint QwtPlotPicker::transform ( const QPointF & pos ) const [protected]`

Translate a point from plot into pixel coordinates

Returns

Point in pixel coordinates

See also

invTransform()

Definition at line 383 of file qwt_plot_picker.cpp.

14.92.3.20 transform() [2/2] QRect QwtPlotPicker::transform (const QRectF & *rect*) const [protected]

Translate a rectangle from plot into pixel coordinates

Returns

Rectangle in pixel coordinates

See also

invTransform()

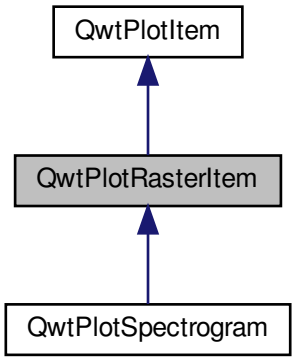
Definition at line 354 of file qwt_plot_picker.cpp.

14.93 QwtPlotRasterItem Class Reference

A class, which displays raster data.

`#include <qwt_plot_rasteritem.h>`

Inheritance diagram for QwtPlotRasterItem:



Public Types

- enum CachePolicy { NoCache , PaintCache }

  Cache policy The default policy is NoCache.
- enum PaintAttribute { PaintInDeviceResolution = 1 }
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPlotRasterItem (const QString &title=QString())

  Constructor.
- QwtPlotRasterItem (const QwtText &title)

  Constructor.
- virtual ~QwtPlotRasterItem ()

  Destructor.
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setAlpha (int alpha)

  Set an alpha value for the raster data.
- int alpha () const
- void setCachePolicy (CachePolicy)
- CachePolicy cachePolicy () const
- void invalidateCache ()
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override

  Draw the raster data.
- virtual QRectF pixelHint (const QRectF &) const

  Pixel hint.
- virtual QwtInterval interval (Qt::Axis) const
- virtual QRectF boundingRect () const override

Protected Member Functions

- virtual QImage renderImage (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &area, const QSize &imageSize) const =0

  Render an image.
- virtual QwtScaleMap imageMap (Qt::Orientation, const QwtScaleMap &map, const QRectF &area, const QSize &imageSize, double pixelSize) const

  Calculate a scale map for painting to an image.

14.93.1 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

Definition at line 37 of file qwt_plot_rasteritem.h.

14.93.2 Member Typedef Documentation

14.93.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotRasterItem::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 83 of file qwt_plot_rasteritem.h.

14.93.3 Member Enumeration Documentation

14.93.3.1 CachePolicy `enum QwtPlotRasterItem::CachePolicy`

Cache policy The default policy is NoCache.

Enumerator

NoCache	renderImage() is called each time 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 useful for improving the performance of hide/show operations or manipulations of the alpha value. All other situations are handled by the canvas backing store.

Definition at line 44 of file qwt_plot_rasteritem.h.

14.93.3.2 PaintAttribute `enum QwtPlotRasterItem::PaintAttribute`

Attributes to modify the drawing algorithm.

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

PaintInDeviceResolution	When the image is rendered according to the data pixels (QwtRasterData::pixelHint()) it can be expanded to paint device resolution before it is passed to QPainter. The expansion algorithm rounds the pixel borders in the same way as the axis ticks, what is usually better than the scaling algorithm implemented in Qt. Disabling this flag might make sense, to reduce the size of a document/file. If this is possible for a document format depends on the implementation of the specific QPaintEngine.

Definition at line 66 of file qwt_plot_rasteritem.h.

14.93.4 Member Function Documentation

14.93.4.1 alpha() `int QwtPlotRasterItem::alpha ( ) const`

Returns

Alpha value of the raster item

See also

setAlpha()

Definition at line 530 of file qwt_plot_rasteritem.cpp.

14.93.4.2 boundingRect() `QRectF QwtPlotRasterItem::boundingRect ( ) const [override], [virtual]`

Returns

Bounding rectangle of the data

See also

QwtPlotRasterItem::interval()

Reimplemented from QwtPlotItem.

Definition at line 788 of file qwt_plot_rasteritem.cpp.

14.93.4.3 cachePolicy() `QwtPlotRasterItem::CachePolicy QwtPlotRasterItem::cachePolicy ( ) const`

Returns

Cache policy

See also

CachePolicy, setCachePolicy()

Definition at line 559 of file qwt_plot_rasteritem.cpp.

14.93.4.4 draw()

```
void QwtPlotRasterItem::draw (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const [override], [virtual]
```

Draw the raster data.

Parameters

painter	Painter
xMap	X-Scale Map
yMap	Y-Scale Map
canvasRect	Contents rectangle of the plot canvas

Implements QwtPlotItem.

Reimplemented in QwtPlotSpectrogram.

Definition at line 614 of file qwt_plot_rasteritem.cpp.

14.93.4.5 imageMap()

```
QwtScaleMap QwtPlotRasterItem::imageMap (
        Qt::Orientation orientation,
        const QwtScaleMap & map,
        const QRectF & area,
        const QSize & imageSize,
        double pixelSize ) const  [protected], [virtual]
```

Calculate a scale map for painting to an image.

Parameters

orientation	Orientation, Qt::Horizontal means a X axis
map	Scale map for rendering the plot item
area	Area to be painted on the image
imageSize	Image size
pixelSize	Width/Height of a data pixel

Returns

Calculated scale map

Definition at line 927 of file qwt_plot_rasteritem.cpp.

14.93.4.6 interval()

```
QwtInterval QwtPlotRasterItem::interval (
        Qt::Axis axis ) const  [virtual]
```

Returns

Bounding interval for an axis

This method is intended to be reimplemented by derived classes. The default implementation returns an invalid interval.

Parameters

axis	X, Y, or Z axis

Reimplemented in QwtPlotSpectrogram.

Definition at line 778 of file qwt_plot_rasteritem.cpp.

14.93.4.7 invalidateCache() `void QwtPlotRasterItem::invalidateCache ( )`

Invalidate the paint cache

See also

setCachePolicy()

Definition at line 568 of file qwt_plot_rasteritem.cpp.

14.93.4.8 pixelHint() `QRectF QwtPlotRasterItem::pixelHint ( const QRectF & area ) const [virtual]`

Pixel hint.

The geometry of a pixel is used to calculated the resolution and alignment of the rendered image.

Width and height of the hint need to be the horizontal and vertical distances between 2 neighbored points. The center of the hint has to be the position of any point (it doesn't matter which one).

Limiting the resolution of the image might significantly improve the performance and heavily reduce the amount of memory when rendering a QImage from the raster data.

The default implementation returns an empty rectangle (QRectF()), meaning, that the image will be rendered in target device (f.e screen) resolution.

Parameters

area	In most implementations the resolution of the data doesn't depend on the requested area.

Returns

Bounding rectangle of a pixel

See also

render(), renderImage()

Reimplemented in QwtPlotSpectrogram.

Definition at line 601 of file qwt_plot_rasteritem.cpp.

14.93.4.9 renderImage()

```
virtual QImage QwtPlotRasterItem::renderImage (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & area,
    const QSize & imageSize ) const  [protected], [pure virtual]
```

Render an image.

An implementation of render() might iterate over all pixels of imageRect. Each pixel has to be translated into the corresponding position in scale coordinates using the maps. This position can be used to look up a value in a implementation specific way and to map it into a color.

Parameters

xMap	X-Scale Map
yMap	Y-Scale Map
area	Requested area for the image in scale coordinates
imageSize	Requested size of the image

Returns

Rendered image

Implemented in QwtPlotSpectrogram.

14.93.4.10 setAlpha()

```
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

alpha	Alpha value

- alpha >= 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()

Definition at line 510 of file qwt_plot_rasteritem.cpp.

14.93.4.11 setCachePolicy() `void QwtPlotRasterItem::setCachePolicy ( QwtPlotRasterItem::CachePolicy policy )`

Change the cache policy

The default policy is NoCache

Parameters

policy	Cache policy

See also

CachePolicy, cachePolicy()

Definition at line 543 of file qwt_plot_rasteritem.cpp.

14.93.4.12 setPaintAttribute() `void QwtPlotRasterItem::setPaintAttribute ( PaintAttribute attribute, bool on = true )`

Specify an attribute how to draw the raster item

Parameters

attribute	Paint attribute
on	On/Off /sa PaintAttribute, testPaintAttribute()

Definition at line 470 of file qwt_plot_rasteritem.cpp.

14.93.4.13 testPaintAttribute() `bool QwtPlotRasterItem::testPaintAttribute ( PaintAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

PaintAttribute, setPaintAttribute()

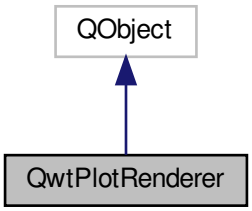
Definition at line 482 of file qwt_plot_rasteritem.cpp.

14.94 QwtPlotRenderer Class Reference

Renderer for exporting a plot to a document, a printer or anything else, that is supported by QPainter/QPaintDevice.

```
#include <qwt_plot_renderer.h>
```

Inheritance diagram for QwtPlotRenderer:



Public Types

- enum DiscardFlag {
  DiscardNone = 0x00 , DiscardBackground = 0x01 , DiscardTitle = 0x02 , DiscardLegend = 0x04 ,
  DiscardCanvasBackground = 0x08 , DiscardFooter = 0x10 , DiscardCanvasFrame = 0x20 }
  Discard flags.
- enum LayoutFlag { DefaultLayout = 0x00 , FrameWithScales = 0x01 }
  Layout flags.
- typedef QFlags< DiscardFlag > DiscardFlags
- typedef QFlags< LayoutFlag > LayoutFlags

Public Member Functions

- QwtPlotRenderer (QObject *=NULL)
- virtual ~QwtPlotRenderer ()

  Destructor.
- void setDiscardFlag (DiscardFlag flag, bool on=true)
- bool testDiscardFlag (DiscardFlag flag) const
- void setDiscardFlags (DiscardFlags flags)
- DiscardFlags discardFlags () const
- void setLayoutFlag (LayoutFlag flag, bool on=true)
- bool testLayoutFlag (LayoutFlag flag) const
- void setLayoutFlags (LayoutFlags flags)
- LayoutFlags layoutFlags () const
- void renderDocument (QwtPlot *, const QString &fileName, const QSizeF &sizeMM, int resolution=85)
- void renderDocument (QwtPlot *, const QString &fileName, const QString &format, const QSizeF &sizeMM, int resolution=85)
- void renderTo (QwtPlot *, QPrinter &) const

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

  Render the plot to a `QPaintDevice`.
- virtual void render (QwtPlot *, QPainter *, const QRectF &plotRect) const
- virtual void renderTitle (const QwtPlot *, QPainter *, const QRectF &titleRect) const
- virtual void renderFooter (const QwtPlot *, QPainter *, const QRectF &footerRect) const
- virtual void renderScale (const QwtPlot *, QPainter *, QwtAxisId, int startDist, int endDist, int baseDist, const QRectF &scaleRect) const

  Paint a scale into a given rectangle. Paint the scale into a given rectangle.
- virtual void renderCanvas (const QwtPlot *, QPainter *, const QRectF &canvasRect, const QwtScaleMap *maps) const
- virtual void renderLegend (const QwtPlot *, QPainter *, const QRectF &legendRect) const
- bool exportTo (QwtPlot *, const QString &documentName, const QSizeF &sizeMM=QSizeF(300, 200), int resolution=85)

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

14.94.1 Detailed Description

Renderer for exporting a plot to a document, a printer or anything else, that is supported by QPainter/QPaintDevice.

Definition at line 39 of file qwt_plot_renderer.h.

14.94.2 Member Typedef Documentation

14.94.2.1 DiscardFlags `typedef QFlags<DiscardFlag > QwtPlotRenderer::DiscardFlags`

An ORed combination of DiscardFlag values.

Definition at line 76 of file qwt_plot_renderer.h.

14.94.2.2 LayoutFlags `typedef QFlags<LayoutFlag > QwtPlotRenderer::LayoutFlags`

An ORed combination of LayoutFlag values.

Definition at line 94 of file qwt_plot_renderer.h.

14.94.3 Member Enumeration Documentation

14.94.3.1 DiscardFlag `enum QwtPlotRenderer::DiscardFlag`

Discard flags.

Enumerator

DiscardNone	Render all components of the plot.
DiscardBackground	Don't render the background of the plot.
DiscardTitle	Don't render the title of the plot.
DiscardLegend	Don't render the legend of the plot.
DiscardCanvasBackground	Don't render the background of the canvas.
DiscardFooter	Don't render the footer of the plot.
DiscardCanvasFrame	Don't render the frame of the canvas **Note** This flag has no effect when using style sheets, where the frame is part of the background

Definition at line 45 of file qwt_plot_renderer.h.

14.94.3.2 LayoutFlag `enum QwtPlotRenderer::LayoutFlag`

Layout flags.

See also

setLayoutFlag(), testLayoutFlag()

Enumerator

DefaultLayout	Use the default layout as on screen.
FrameWithScales	Instead of the scales a box is painted around the plot canvas, where the scale ticks are aligned to.

Definition at line 82 of file qwt_plot_renderer.h.

14.94.4 Constructor & Destructor Documentation

14.94.4.1 QwtPlotRenderer() `QwtPlotRenderer::QwtPlotRenderer ( QObject * parent = NULL ) [explicit]`

Constructor

Parameters

parent	Parent object

Definition at line 146 of file qwt_plot_renderer.cpp.

14.94.5 Member Function Documentation

14.94.5.1 discardFlags() `QwtPlotRenderer::DiscardFlags QwtPlotRenderer::discardFlags ( ) const`

Returns

Flags, indicating what to discard from rendering

See also

DiscardFlag, setDiscardFlags(), setDiscardFlag(), testDiscardFlag()

Definition at line 199 of file qwt_plot_renderer.cpp.

14.94.5.2 exportTo()

```
bool QwtPlotRenderer::exportTo (
    QwtPlot * plot,
    const QString & documentName,
    const QSizeF & sizeMM = QSizeF( 300, 200 ),
    int resolution = 85 )
```

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

Parameters

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

Returns

True, when exporting was successful

See also

renderDocument()

Definition at line 1060 of file qwt_plot_renderer.cpp.

14.94.5.3 layoutFlags() `QwtPlotRenderer::LayoutFlags QwtPlotRenderer::layoutFlags ( ) const`

Returns

Layout flags

See also

LayoutFlag, setLayoutFlags(), setLayoutFlag(), testLayoutFlag()

Definition at line 245 of file qwt_plot_renderer.cpp.

14.94.5.4 render()

```
void QwtPlotRenderer::render (
        QwtPlot * plot,
        QPainter * painter,
        const QRectF & plotRect ) const  [virtual]
```

Paint the contents of a QwtPlot instance into a given rectangle.

Parameters

plot	Plot to be rendered
painter	Painter
plotRect	Bounding rectangle

See also

renderDocument(), renderTo(), QwtPainter::setRoundingAlignment()

Definition at line 482 of file qwt_plot_renderer.cpp.

14.94.5.5 renderCanvas()

```
void QwtPlotRenderer::renderCanvas (
    const QwtPlot * plot,
    QPainter * painter,
    const QRectF & canvasRect,
    const QwtScaleMap * maps ) const  [virtual]
```

Render the canvas into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
maps	Maps mapping between plot and paint device coordinates
canvasRect	Canvas rectangle

Definition at line 831 of file qwt_plot_renderer.cpp.

14.94.5.6 renderDocument() [1/2]

```
void QwtPlotRenderer::renderDocument (
    QwtPlot * plot,
    const QString & fileName,
    const QSizeF & sizeMM,
    int resolution = 85 )
```

Render a plot to a file

The format of the document will be auto-detected from the suffix of the file name.

Parameters

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

Definition at line 261 of file qwt_plot_renderer.cpp.

14.94.5.7 renderDocument() [2/2]

```
void QwtPlotRenderer::renderDocument (
    QwtPlot * plot,
    const QString & fileName,
    const QString & format,
    const QSizeF & sizeMM,
    int resolution = 85 )
```

Render a plot to a file

Supported formats are:

- pdf
  Portable Document Format PDF
- ps
  Postcript
- svg
  Scalable Vector Graphics SVG
- all image formats supported by Qt
  see QImageWriter::supportedImageFormats()

Scalable vector graphic formats like PDF or SVG are superior to raster graphics formats.

Parameters

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

See also

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

Definition at line 293 of file qwt_plot_renderer.cpp.

14.94.5.8 renderFooter()

```
void QwtPlotRenderer::renderFooter (
    const QwtPlot * plot,
    QPainter * painter,
    const QRectF & footerRect ) const  [virtual]
```

Render the footer into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
footerRect	Bounding rectangle for the footer

Definition at line 692 of file qwt_plot_renderer.cpp.

14.94.5.9 renderLegend()

```
void QwtPlotRenderer::renderLegend (
    const QwtPlot * plot,
    QPainter * painter,
    const QRectF & legendRect ) const  [virtual]
```

Render the legend into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
legendRect	Bounding rectangle for the legend

Definition at line 711 of file qwt_plot_renderer.cpp.

14.94.5.10 renderScale()

```
void QwtPlotRenderer::renderScale (
    const QwtPlot * plot,
    QPainter * painter,
    QwtAxisId axisId,
    int startDist,
    int endDist,
    int baseDist,
    const QRectF & scaleRect ) const  [virtual]
```

Paint a scale into a given rectangle. Paint the scale into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
axisId	Axis
startDist	Start border distance
endDist	End border distance
baseDist	Base distance
scaleRect	Bounding rectangle for the scale

Definition at line 733 of file qwt_plot_renderer.cpp.

14.94.5.11 renderTitle()

```
void QwtPlotRenderer::renderTitle (
    const QwtPlot * plot,
    QPainter * painter,
    const QRectF & titleRect ) const  [virtual]
```

Render the title into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
titleRect	Bounding rectangle for the title

Definition at line 673 of file qwt_plot_renderer.cpp.

14.94.5.12 renderTo() [1/2] void QwtPlotRenderer::renderTo (QwtPlot * *plot*, QPaintDevice & *paintDevice*) const

Render the plot to a QPaintDevice.

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

Parameters

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

See also

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

Definition at line 402 of file qwt_plot_renderer.cpp.

14.94.5.13 renderTo() [2/2] void QwtPlotRenderer::renderTo (QwtPlot * *plot*, QPrinter & *printer*) const

Render the plot to a QPrinter.

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

Parameters

plot	Plot to be rendered
printer	Printer to paint on

See also

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

Definition at line 427 of file qwt_plot_renderer.cpp.

14.94.5.14 setDiscardFlag() void QwtPlotRenderer::setDiscardFlag (DiscardFlag *flag*, bool *on* = *true*)

Change a flag, indicating what to discard from rendering

Parameters

flag	Flag to change
on	On/Off

See also

DiscardFlag, testDiscardFlag(), setDiscardFlags(), discardFlags()

Definition at line 166 of file qwt_plot_renderer.cpp.

14.94.5.15 setDiscardFlags()

```
void QwtPlotRenderer::setDiscardFlags (
        DiscardFlags flags )
```

Set the flags, indicating what to discard from rendering

Parameters

flags	Flags

See also

DiscardFlag, setDiscardFlag(), testDiscardFlag(), discardFlags()

Definition at line 190 of file qwt_plot_renderer.cpp.

14.94.5.16 setLayoutFlag()

```
void QwtPlotRenderer::setLayoutFlag (
        LayoutFlag flag,
        bool on = true )
```

Change a layout flag

Parameters

flag	Flag to change
on	On/Off

See also

LayoutFlag, testLayoutFlag(), setLayoutFlags(), layoutFlags()

Definition at line 212 of file qwt_plot_renderer.cpp.

14.94.5.17 setLayoutFlags() `void QwtPlotRenderer::setLayoutFlags ( LayoutFlags flags )`

Set the layout flags

Parameters

flags	Flags

See also

LayoutFlag, setLayoutFlag(), testLayoutFlag(), layoutFlags()

Definition at line 236 of file qwt_plot_renderer.cpp.

14.94.5.18 testDiscardFlag() `bool QwtPlotRenderer::testDiscardFlag ( DiscardFlag flag ) const`

Returns

True, if flag is enabled.

Parameters

flag	Flag to be tested

See also

DiscardFlag, setDiscardFlag(), setDiscardFlags(), discardFlags()

Definition at line 179 of file qwt_plot_renderer.cpp.

14.94.5.19 testLayoutFlag() `bool QwtPlotRenderer::testLayoutFlag ( LayoutFlag flag ) const`

Returns

True, if flag is enabled.

Parameters

flag	Flag to be tested

See also

LayoutFlag, setLayoutFlag(), setLayoutFlags(), layoutFlags()

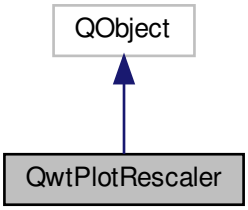
Definition at line 225 of file qwt_plot_renderer.cpp.

14.95 QwtPlotRescaler Class Reference

QwtPlotRescaler takes care of fixed aspect ratios for plot scales.

`#include <qwt_plot_rescaler.h>`

Inheritance diagram for QwtPlotRescaler:



Public Types

- enum RescalePolicy { Fixed , Expanding , Fitting }
- enum ExpandingDirection { ExpandUp , ExpandDown , ExpandBoth }

Public Member Functions

- QwtPlotRescaler (QWidget *canvas, QwtAxisId referenceAxis=QwtAxis::XBottom, RescalePolicy=Expanding)
- virtual ~QwtPlotRescaler ()

  Destructor.
- void setEnabled (bool)

  En/disable the rescaler.
- bool isEnabled () const
- void setRescalePolicy (RescalePolicy)
- RescalePolicy rescalePolicy () const
- void setExpandingDirection (ExpandingDirection)
- void setExpandingDirection (QwtAxisId, ExpandingDirection)
- ExpandingDirection expandingDirection (QwtAxisId) const
- void setReferenceAxis (QwtAxisId)
- QwtAxisId referenceAxis () const
- void setAspectRatio (double ratio)
- void setAspectRatio (QwtAxisId, double ratio)
- double aspectRatio (QwtAxisId) const
- void setIntervalHint (QwtAxisId, const QwtInterval &)

- QwtInterval intervalHint (QwtAxisId) const
- QWidget * canvas ()
- const QWidget * canvas () const
- QwtPlot * plot ()
- const QwtPlot * plot () const
- virtual bool eventFilter (QObject *, QEvent *) override

  Event filter for the plot canvas.
- void rescale () const

  Adjust the plot axes scales.

Protected Member Functions

- virtual void canvasResizeEvent (QResizeEvent *)
- virtual void rescale (const QSize &oldSize, const QSize &newSize) const
- virtual QwtInterval expandScale (QwtAxisId, const QSize &oldSize, const QSize &newSize) const
- virtual QwtInterval syncScale (QwtAxisId, const QwtInterval &reference, const QSize &size) const
- virtual void updateScales (QwtInterval intervals[QwtAxis::AxisPositions]) const
- Qt::Orientation orientation (QwtAxisId) const
- QwtInterval interval (QwtAxisId) const
- QwtInterval expandInterval (const QwtInterval &, double width, ExpandingDirection) const

14.95.1 Detailed Description

QwtPlotRescaler takes care of fixed aspect ratios for plot scales.

QwtPlotRescaler auto adjusts the axes of a QwtPlot according to fixed aspect ratios.

Definition at line 29 of file qwt_plot_rescaler.h.

14.95.2 Member Enumeration Documentation

14.95.2.1 ExpandingDirection `enum QwtPlotRescaler::ExpandingDirection`

When rescalePolicy() is set to Expanding its direction depends on ExpandingDirection

Enumerator

ExpandUp	The upper limit of the scale is adjusted.
ExpandDown	The lower limit of the scale is adjusted.
ExpandBoth	Both limits of the scale are adjusted.

Definition at line 70 of file qwt_plot_rescaler.h.

14.95.2.2 RescalePolicy `enum QwtPlotRescaler::RescalePolicy`

The rescale policy defines how to rescale the reference axis and their depending axes.

See also

ExpandingDirection, setIntervalHint()

Enumerator

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 interval hint.

Definition at line 40 of file qwt_plot_rescaler.h.

14.95.3 Constructor & Destructor Documentation

14.95.3.1 QwtPlotRescaler()

```
QwtPlotRescaler::QwtPlotRescaler (
        QWidget * canvas,
        QwtAxisId referenceAxis = QwtAxis::XBottom,
        RescalePolicy policy = Expanding )  [explicit]
```

Constructor

Parameters

canvas	Canvas
referenceAxis	Reference axis, see RescalePolicy
policy	Rescale policy

See also

setRescalePolicy(), setReferenceAxis()

Definition at line 71 of file qwt_plot_rescaler.cpp.

14.95.4 Member Function Documentation

14.95.4.1 aspectRatio() `double QwtPlotRescaler::aspectRatio ( QwtAxisId axisId ) const`

Returns

Aspect ratio between an axis and the reference axis.

Parameters

axisId	Axis

See also

setAspectRatio()

Definition at line 243 of file qwt_plot_rescaler.cpp.

14.95.4.2 canvas() [1/2] `QWidget * QwtPlotRescaler::canvas ( )`

Returns

plot canvas

Definition at line 282 of file qwt_plot_rescaler.cpp.

14.95.4.3 canvas() [2/2] `const QWidget * QwtPlotRescaler::canvas ( ) const`

Returns

plot canvas

Definition at line 288 of file qwt_plot_rescaler.cpp.

14.95.4.4 canvasResizeEvent() `void QwtPlotRescaler::canvasResizeEvent ( QResizeEvent * event ) [protected], [virtual]`

Event handler for resize events of the plot canvas

Parameters

event	Resize event

See also

rescale()

Definition at line 343 of file qwt_plot_rescaler.cpp.

14.95.4.5 expandingDirection()

```
QwtPlotRescaler::ExpandingDirection QwtPlotRescaler::expandingDirection (
            QwtAxisId axisId ) const
```

Returns

Direction in which an axis should be expanded

Parameters

axisId	Axis

See also

setExpandingDirection()

Definition at line 197 of file qwt_plot_rescaler.cpp.

14.95.4.6 expandInterval()

```
QwtInterval QwtPlotRescaler::expandInterval (
            const QwtInterval & interval,
            double width,
            ExpandingDirection direction ) const  [protected]
```

Expand the interval

Parameters

interval	Interval to be expanded
width	Distance to be added to the interval
direction	Direction of the expand operation

Returns

Expanded interval

Definition at line 526 of file qwt_plot_rescaler.cpp.

14.95.4.7 expandScale()

```
QwtInterval QwtPlotRescaler::expandScale (
        QwtAxisId axisId,
        const QSize & oldSize,
        const QSize & newSize ) const  [protected], [virtual]
```

Calculate the new scale interval of a plot axis

Parameters

axisId	Axis
oldSize	Previous size of the canvas
newSize	New size of the canvas

Returns

Calculated new interval for the axis

Definition at line 405 of file qwt_plot_rescaler.cpp.

14.95.4.8 interval()

```
QwtInterval QwtPlotRescaler::interval (
        QwtAxisId axisId ) const  [protected]
```

Parameters

axisId	Axis

Returns

Normalized interval of an axis

Definition at line 509 of file qwt_plot_rescaler.cpp.

14.95.4.9 intervalHint()

```
QwtInterval QwtPlotRescaler::intervalHint (
        QwtAxisId axisId ) const
```

Parameters

axisId	Axis

Returns

Interval hint

See also

setIntervalHint(), RescalePolicy

Definition at line 273 of file qwt_plot_rescaler.cpp.

14.95.4.10 isEnabled() `bool QwtPlotRescaler::isEnabled ( ) const`

Returns

true when enabled, false otherwise

See also

setEnabled, eventFilter()

Definition at line 118 of file qwt_plot_rescaler.cpp.

14.95.4.11 orientation() `Qt::Orientation QwtPlotRescaler::orientation ( QwtAxisId axisId ) const [protected]`

Returns

Orientation of an axis

Parameters

axisId	Axis

Definition at line 500 of file qwt_plot_rescaler.cpp.

14.95.4.12 plot() [1/2] `QwtPlot * QwtPlotRescaler::plot ( )`

Returns

plot widget

Definition at line 294 of file qwt_plot_rescaler.cpp.

14.95.4.13 plot() [2/2] const QwtPlot * QwtPlotRescaler::plot ( ) const

Returns

plot widget

Definition at line 304 of file qwt_plot_rescaler.cpp.

14.95.4.14 referenceAxis() QwtAxisId QwtPlotRescaler::referenceAxis ( ) const

Returns

Reference axis (see RescalePolicy)

See also

setReferenceAxis()

Definition at line 158 of file qwt_plot_rescaler.cpp.

14.95.4.15 rescale() void QwtPlotRescaler::rescale (const QSize & *oldSize*, const QSize & *newSize*) const [protected], [virtual]

Adjust the plot axes scales

Parameters

oldSize	Previous size of the canvas
newSize	New size of the canvas

Definition at line 367 of file qwt_plot_rescaler.cpp.

14.95.4.16 rescalePolicy() QwtPlotRescaler::RescalePolicy QwtPlotRescaler::rescalePolicy ( ) const

Returns

Rescale policy

See also

setRescalePolicy()

Definition at line 138 of file qwt_plot_rescaler.cpp.

14.95.4.17 setAspectRatio() [1/2] 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

ratio	Aspect ratio

See also

aspectRatio()

Definition at line 212 of file qwt_plot_rescaler.cpp.

14.95.4.18 setAspectRatio() [2/2] void QwtPlotRescaler::setAspectRatio (QwtAxisId *axisId*, double *ratio*)

Set the aspect ratio between the scale of the reference axis and another scale. The default ratio is 1.0

Parameters

axisId	Axis
ratio	Aspect ratio

See also

aspectRatio()

Definition at line 226 of file qwt_plot_rescaler.cpp.

14.95.4.19 setEnabled() 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

on	true or false

See also

isEnabled(), eventFilter()

Definition at line 97 of file qwt_plot_rescaler.cpp.

14.95.4.20 setExpandingDirection() [1/2] void QwtPlotRescaler::setExpandingDirection (ExpandingDirection *direction*)

Set the direction in which all axis should be expanded

Parameters

direction	Direction

See also

expandingDirection()

Definition at line 169 of file qwt_plot_rescaler.cpp.

14.95.4.21 setExpandingDirection() [2/2] void QwtPlotRescaler::setExpandingDirection (QwtAxisId *axisId*, ExpandingDirection *direction*)

Set the direction in which an axis should be expanded

Parameters

axisId	Axis
direction	Direction

See also

expandingDirection()

Definition at line 183 of file qwt_plot_rescaler.cpp.

14.95.4.22 setIntervalHint() void QwtPlotRescaler::setIntervalHint (QwtAxisId *axisId*, const QwtInterval & *interval*)

Set an interval hint for an axis

In Fitting mode, the hint is used as minimal interval that always needs to be displayed.

Parameters

axisId	Axis
interval	Axis

See also

intervalHint(), RescalePolicy

Definition at line 261 of file qwt_plot_rescaler.cpp.

14.95.4.23 setReferenceAxis() `void QwtPlotRescaler::setReferenceAxis ( QwtAxisId axisId )`

Set the reference axis (see RescalePolicy)

Parameters

axisId	Axis

See also

referenceAxis()

Definition at line 149 of file qwt_plot_rescaler.cpp.

14.95.4.24 setRescalePolicy() `void QwtPlotRescaler::setRescalePolicy ( RescalePolicy policy )`

Change the rescale policy

Parameters

policy	Rescale policy

See also

rescalePolicy()

Definition at line 129 of file qwt_plot_rescaler.cpp.

14.95.4.25 syncScale()

```
QwtInterval QwtPlotRescaler::syncScale (
    QwtAxisId axisId,
    const QwtInterval & reference,
    const QSize & size ) const  [protected], [virtual]
```

Synchronize an axis scale according to the scale of the reference axis

Parameters

axisId	Axis
reference	Interval of the reference axis
size	Size of the canvas

Returns

New interval for axis

Definition at line 469 of file qwt_plot_rescaler.cpp.

14.95.4.26 updateScales()

```
void QwtPlotRescaler::updateScales (
    QwtInterval intervals[QwtAxis::AxisPositions] ) const  [protected], [virtual]
```

Update the axes scales

Parameters

intervals	Scale intervals

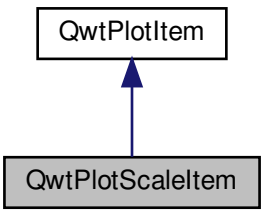
Definition at line 588 of file qwt_plot_rescaler.cpp.

14.96 QwtPlotScaleItem Class Reference

A class which draws a scale inside the plot canvas.

`#include <qwt_plot_scaleitem.h>`

Inheritance diagram for QwtPlotScaleItem:



Public Member Functions

- QwtPlotScaleItem (QwtScaleDraw::Alignment=QwtScaleDraw::BottomScale, const double pos=0.0)

  Constructor for scale item at the position pos.
- virtual ~QwtPlotScaleItem ()

  Destructor.
- virtual int rtti () const override
- void setScaleDiv (const QwtScaleDiv &)

  Assign a scale division.
- const QwtScaleDiv & scaleDiv () const
- void setScaleDivFromAxis (bool on)
- bool isScaleDivFromAxis () const
- void setPalette (const QPalette &)
- QPalette palette () const
- void setFont (const QFont &)
- QFont font () const
- void setScaleDraw (QwtScaleDraw *)

  Set a scale draw.
- const QwtScaleDraw * scaleDraw () const
- QwtScaleDraw * scaleDraw ()
- void setPosition (double pos)
- double position () const
- void setBorderDistance (int)

  Align the scale to the canvas.
- int borderDistance () const
- void setAlignment (QwtScaleDraw::Alignment)
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override

  Draw the scale.
- virtual void updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &) override

  Update the item to changes of the axes scale division.

Additional Inherited Members

14.96.1 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 by 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( QwtAxis::YLeft )->font() );
scaleItem->attach(plot);
plot->setAxisVisible( QwtAxis::YLeft, false );
```

Definition at line 46 of file qwt_plot_scaleitem.h.

14.96.2 Constructor & Destructor Documentation

14.96.2.1 QwtPlotScaleItem() `QwtPlotScaleItem::QwtPlotScaleItem ( QwtScaleDraw::Alignment alignment = QwtScaleDraw::BottomScale, const double pos = 0.0 ) [explicit]`

Constructor for scale item at the position pos.

Parameters

alignment	In case of QwtScaleDraw::BottomScale or QwtScaleDraw::TopScale the scale item is corresponding to the xAxis(), otherwise it corresponds to the yAxis().
pos	x or y position, depending on the corresponding axis.

See also

setPosition(), setAlignment()

Definition at line 75 of file qwt_plot_scaleitem.cpp.

14.96.3 Member Function Documentation

14.96.3.1 borderDistance() `int QwtPlotScaleItem::borderDistance ( ) const`

Returns

Distance from a canvas border

See also

setBorderDistance(), setPosition()

Definition at line 312 of file qwt_plot_scaleitem.cpp.

14.96.3.2 font() `QFont QwtPlotScaleItem::font ( ) const`

Returns

tick label font

See also

setFont()

Definition at line 196 of file qwt_plot_scaleitem.cpp.

14.96.3.3 isScaleDivFromAxis() `bool QwtPlotScaleItem::isScaleDivFromAxis ( ) const`

Returns

True, if the synchronization of the scale division with the corresponding axis is enabled.

See also

setScaleDiv(), setScaleDivFromAxis()

Definition at line 150 of file qwt_plot_scaleitem.cpp.

14.96.3.4 palette() `QPalette QwtPlotScaleItem::palette ( ) const`

Returns

palette

See also

setPalette()

Definition at line 174 of file qwt_plot_scaleitem.cpp.

14.96.3.5 position() `double QwtPlotScaleItem::position ( ) const`

Returns

Position of the scale

See also

setPosition(), setAlignment()

Definition at line 275 of file qwt_plot_scaleitem.cpp.

14.96.3.6 rtti() `int QwtPlotScaleItem::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotScale

Reimplemented from QwtPlotItem.

Definition at line 94 of file qwt_plot_scaleitem.cpp.

14.96.3.7 scaleDiv() const QwtScaleDiv & QwtPlotScaleItem::scaleDiv ( ) const

Returns

Scale division

Definition at line 115 of file qwt_plot_scaleitem.cpp.

14.96.3.8 scaleDraw() [1/2] QwtScaleDraw * QwtPlotScaleItem::scaleDraw ( )

Returns

Scale draw

See also

setScaleDraw()

Definition at line 245 of file qwt_plot_scaleitem.cpp.

14.96.3.9 scaleDraw() [2/2] const QwtScaleDraw * QwtPlotScaleItem::scaleDraw ( ) const

Returns

Scale draw

See also

setScaleDraw()

Definition at line 236 of file qwt_plot_scaleitem.cpp.

14.96.3.10 setAlignment() 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()

Definition at line 333 of file qwt_plot_scaleitem.cpp.

14.96.3.11 setBorderDistance() `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 rectangle 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

distance	Number of pixels between the canvas border and the backbone of the scale.

See also

setPosition(), borderDistance()

Definition at line 296 of file qwt_plot_scaleitem.cpp.

14.96.3.12 setFont() `void QwtPlotScaleItem::setFont ( const QFont & font )`

Change the tick label font

See also

font()

Definition at line 183 of file qwt_plot_scaleitem.cpp.

14.96.3.13 setPalette() `void QwtPlotScaleItem::setPalette ( const QPalette & palette )`

Set the palette

See also

QwtAbstractScaleDraw::draw(), palette()

Definition at line 159 of file qwt_plot_scaleitem.cpp.

14.96.3.14 setPosition() `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

pos	New position

See also

position(), setAlignment()

Definition at line 261 of file qwt_plot_scaleitem.cpp.

14.96.3.15 setScaleDiv() `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

scaleDiv	Scale division

See also

scaleDiv(), setScaleDivFromAxis(), isScaleDivFromAxis()

Definition at line 108 of file qwt_plot_scaleitem.cpp.

14.96.3.16 setScaleDivFromAxis() `void QwtPlotScaleItem::setScaleDivFromAxis ( bool on )`

Enable/Disable the synchronization of the scale division with the corresponding axis.

Parameters

on	true/false

See also

isScaleDivFromAxis()

Definition at line 127 of file qwt_plot_scaleitem.cpp.

14.96.3.17 setScaleDraw() void QwtPlotScaleItem::setScaleDraw (QwtScaleDraw * *scaleDraw*)

Set a scale draw.

Parameters

scaleDraw	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()

Definition at line 212 of file qwt_plot_scaleitem.cpp.

14.96.3.18 updateScaleDiv()

```
void QwtPlotScaleItem::updateScaleDiv (
    const QwtScaleDiv & xScaleDiv,
    const QwtScaleDiv & yScaleDiv ) [override], [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

xScaleDiv	Scale division of the x-axis
yScaleDiv	Scale division of the y-axis

See also

QwtPlot::updateAxes()

Reimplemented from QwtPlotItem.

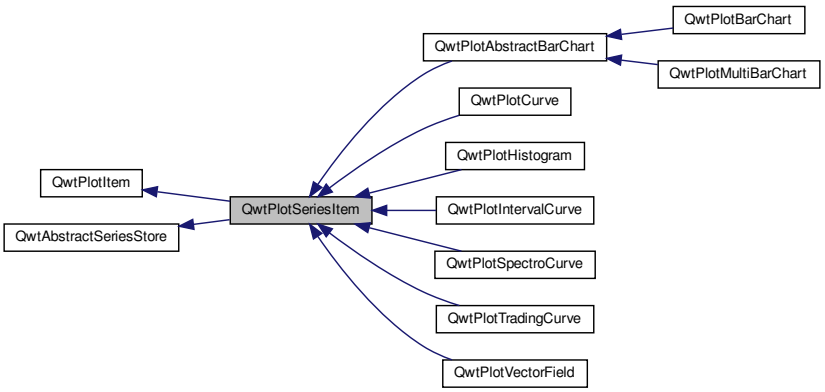
Definition at line 445 of file qwt_plot_scaleitem.cpp.

14.97 QwtPlotSeriesItem Class Reference

Base class for plot items representing a series of samples.

```
#include <qwt_plot_seriesitem.h>
```

Inheritance diagram for QwtPlotSeriesItem:



Public Member Functions

- QwtPlotSeriesItem (const QString &title=QString())
- QwtPlotSeriesItem (const QwtText &title)
- virtual ~QwtPlotSeriesItem ()

  Destructor.
- void setOrientation (Qt::Orientation)
- Qt::Orientation orientation () const
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override

  Draw the complete series.
- virtual void drawSeries (QPainter *painter, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const =0
- virtual QRectF boundingRect () const override
- virtual void updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &) override

  Update the item to changes of the axes scale division.

Protected Member Functions

- virtual void dataChanged () override

  dataChanged() indicates, that the series has been changed.

Additional Inherited Members

14.97.1 Detailed Description

Base class for plot items representing a series of samples.

Definition at line 24 of file qwt_plot_seriesitem.h.

14.97.2 Constructor & Destructor Documentation

14.97.2.1 QwtPlotSeriesItem() [1/2] QwtPlotSeriesItem::QwtPlotSeriesItem (const QString & *title* = *QString()*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 40 of file qwt_plot_seriesitem.cpp.

14.97.2.2 QwtPlotSeriesItem() [2/2]

```
QwtPlotSeriesItem::QwtPlotSeriesItem (
        const QwtText & title ) [explicit]
```

Constructor

Parameters

title	Title of the curve

Definition at line 29 of file qwt_plot_seriesitem.cpp.

14.97.3 Member Function Documentation

14.97.3.1 boundingRect()

```
QRectF QwtPlotSeriesItem::boundingRect ( ) const [override], [virtual]
```

Returns

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

Note

A width or height $<$ 0.0 is ignored by the autoscaler

Reimplemented from QwtPlotItem.

Reimplemented in QwtPlotVectorField, QwtPlotTradingCurve, QwtPlotMultiBarChart, QwtPlotIntervalCurve, QwtPlotHistogram, and QwtPlotBarChart.

Definition at line 97 of file qwt_plot_seriesitem.cpp.

14.97.3.2 draw()

```
void QwtPlotSeriesItem::draw (
        QPainter * painter,
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect ) const [override], [virtual]
```

Draw the complete series.

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas

Implements QwtPlotItem.

Definition at line 90 of file qwt_plot_seriesitem.cpp.

14.97.3.3 drawSeries()

```
virtual void QwtPlotSeriesItem::drawSeries (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [pure virtual]
```

Draw a subset of the samples

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

Implemented in QwtPlotVectorField, QwtPlotTradingCurve, QwtPlotSpectroCurve, QwtPlotMultiBarChart, QwtPlotIntervalCurve, QwtPlotHistogram, QwtPlotCurve, and QwtPlotBarChart.

14.97.3.4 orientation()

```
Qt::Orientation QwtPlotSeriesItem::orientation ( ) const
```

Returns

Orientation of the plot item

See also

setOrientation()

Definition at line 77 of file qwt_plot_seriesitem.cpp.

14.97.3.5 setOrientation() `void QwtPlotSeriesItem::setOrientation ( Qt::Orientation orientation )`

Set the orientation of the item.

The orientation() might be used in specific way by a plot item. F.e. a QwtPlotCurve uses it to identify how to display the curve int QwtPlotCurve::Steps or QwtPlotCurve::Sticks style.

See also

orientation()

Definition at line 62 of file qwt_plot_seriesitem.cpp.

14.97.3.6 updateScaleDiv() `void QwtPlotSeriesItem::updateScaleDiv ( const QwtScaleDiv & xScaleDiv, const QwtScaleDiv & yScaleDiv ) [override], [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()

updateScaleDiv() is only called when the ScaleInterest interest is enabled. The default implementation does nothing.

Parameters

xScaleDiv	Scale division of the x-axis
yScaleDiv	Scale division of the y-axis

See also

QwtPlot::updateAxes(), ScaleInterest

Reimplemented from QwtPlotItem.

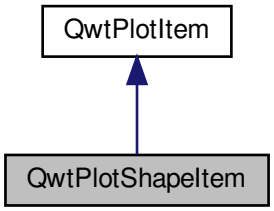
Definition at line 102 of file qwt_plot_seriesitem.cpp.

14.98 QwtPlotShapeItem Class Reference

A plot item, which displays any graphical shape, that can be defined by a QPainterPath.

`#include <qwt_plot_shapeitem.h>`

Inheritance diagram for QwtPlotShapeItem:



Public Types

- enum PaintAttribute { ClipPolygons = 0x01 }
- enum LegendMode { LegendShape , LegendColor }

  Mode how to display the item on the legend.
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPlotShapeItem (const QString &title=QString())

  Constructor.
- QwtPlotShapeItem (const QwtText &title)

  Constructor.
- virtual ~QwtPlotShapeItem ()

  Destructor.
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setLegendMode (LegendMode)
- LegendMode legendMode () const
- void setRect (const QRectF &)

  Set a path built from a rectangle.
- void setPolygon (const QPolygonF &)

  Set a path built from a polygon.
- void setShape (const QPainterPath &)

  Set the shape to be displayed.
- QPainterPath shape () const
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)

  Assign a pen.
- QPen pen () const
- void setBrush (const QBrush &)
- QBrush brush () const
- void setRenderTolerance (double)

  Set the tolerance for the weeding optimization.
- double renderTolerance () const

- virtual QRectF boundingRect () const override

  Bounding rectangle of the shape.
- virtual void draw (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override
- virtual int rtti () const override

Additional Inherited Members

14.98.1 Detailed Description

A plot item, which displays any graphical shape, that can be defined by a QPainterPath.

A QPainterPath is a shape composed from intersecting and uniting regions, rectangles, ellipses or irregular areas defined by lines, and curves. QwtPlotShapeItem displays a shape with a pen and brush.

QwtPlotShapeItem offers a couple of optimizations like clipping or weeding. These algorithms need to convert the painter path into polygons that might be less performant for paths built from curves and ellipses.

More complex shapes, that can't be expressed by a QPainterPath can be displayed using QwtPlotGraphicItem.

See also

QwtPlotZone, QwtPlotGraphicItem

Definition at line 38 of file qwt_plot_shapeitem.h.

14.98.2 Member Typedef Documentation

14.98.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotShapeItem::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 61 of file qwt_plot_shapeitem.h.

14.98.3 Member Enumeration Documentation

14.98.3.1 LegendMode `enum QwtPlotShapeItem::LegendMode`

Mode how to display the item on the legend.

Enumerator

LegendShape	Display a scaled down version of the shape.
LegendColor	Display a filled rectangle.

Definition at line 64 of file qwt_plot_shapeitem.h.

14.98.3.2 PaintAttribute `enum QwtPlotShapeItem::PaintAttribute`

Attributes to modify the drawing algorithm. The default disables all attributes

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

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 But polygon clipping will convert the painter path into polygons what might introduce a negative impact on the performance of paths composed from curves or ellipses.

Definition at line 47 of file qwt_plot_shapeitem.h.

14.98.4 Constructor & Destructor Documentation

14.98.4.1 QwtPlotShapeItem() `[1/2]` `QwtPlotShapeItem::QwtPlotShapeItem ( const QString & title = QString() ) [explicit]`

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

title	Title

Definition at line 112 of file qwt_plot_shapeitem.cpp.

14.98.4.2 QwtPlotShapeItem() `[2/2]` `QwtPlotShapeItem::QwtPlotShapeItem ( const QwtText & title ) [explicit]`

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

title	Title

Definition at line 127 of file qwt_plot_shapeitem.cpp.

14.98.5 Member Function Documentation

14.98.5.1 brush() `QBrush QwtPlotShapeItem::brush ( ) const`

Returns

Brush used to fill the shape

See also

setBrush(), pen()

Definition at line 336 of file qwt_plot_shapeitem.cpp.

14.98.5.2 draw()

```
void QwtPlotShapeItem::draw (
        QPainter * painter,
        const QwtScaleMap & xMap,
        const QwtScaleMap & yMap,
        const QRectF & canvasRect ) const  [override], [virtual]
```

Draw the shape item

Parameters

painter	Painter
xMap	X-Scale Map
yMap	Y-Scale Map
canvasRect	Contents rect of the plot canvas

Implements QwtPlotItem.

Definition at line 385 of file qwt_plot_shapeitem.cpp.

14.98.5.3 legendIcon()

```
QwtGraphic QwtPlotShapeItem::legendIcon (
        int index,
        const QSizeF & size ) const  [override], [virtual]
```

Returns

A rectangle filled with the color of the brush (or the pen)

Parameters

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

See also

setLegendIconSize(), legendData()

Reimplemented from QwtPlotItem.

Definition at line 463 of file qwt_plot_shapeitem.cpp.

14.98.5.4 legendMode()

```
QwtPlotShapeItem::LegendMode QwtPlotShapeItem::legendMode ( ) const
```

Returns

Mode how to represent the item on the legend

See also

legendMode()

Definition at line 199 of file qwt_plot_shapeitem.cpp.

14.98.5.5 pen()

```
QPen QwtPlotShapeItem::pen ( ) const
```

Returns

Pen used to draw the outline of the shape

See also

setPen(), brush()

Definition at line 310 of file qwt_plot_shapeitem.cpp.

14.98.5.6 renderTolerance() `double QwtPlotShapeItem::renderTolerance ( ) const`

Returns

Tolerance for the weeding optimization

See also

setRenderTolerance()

Definition at line 372 of file qwt_plot_shapeitem.cpp.

14.98.5.7 rtti() `int QwtPlotShapeItem::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotShape

Reimplemented from QwtPlotItem.

Definition at line 151 of file qwt_plot_shapeitem.cpp.

14.98.5.8 setBrush() `void QwtPlotShapeItem::setBrush ( const QBrush & brush )`

Assign a brush.

The brush is used to fill the path

Parameters

brush	Brush

See also

brush(), pen()

Definition at line 323 of file qwt_plot_shapeitem.cpp.

14.98.5.9 setLegendMode() `void QwtPlotShapeItem::setLegendMode ( LegendMode mode )`

Set the mode how to represent the item on the legend

Parameters

mode	Mode

See also

legendMode()

Definition at line 186 of file qwt_plot_shapeitem.cpp.

14.98.5.10 setPaintAttribute()

```
void QwtPlotShapeItem::setPaintAttribute (
    PaintAttribute attribute,
    bool on = true )
```

Specify an attribute how to draw the shape

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 163 of file qwt_plot_shapeitem.cpp.

14.98.5.11 setPen() [1/2]

```
void QwtPlotShapeItem::setPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 284 of file qwt_plot_shapeitem.cpp.

14.98.5.12 setPen() [2/2] void QwtPlotShapeItem::setPen (const QPen & *pen*)

Assign a pen.

The pen is used to draw the outline of the shape

Parameters

pen	Pen

See also

pen(), brush()

Definition at line 297 of file qwt_plot_shapeitem.cpp.

14.98.5.13 setPolygon() void QwtPlotShapeItem::setPolygon (const QPolygonF & *polygon*)

Set a path built from a polygon.

Parameters

polygon	Polygon

See also

setShape(), setRect(), shape()

Definition at line 230 of file qwt_plot_shapeitem.cpp.

14.98.5.14 setRect() void QwtPlotShapeItem::setRect (const QRectF & *rect*)

Set a path built from a rectangle.

Parameters

rect	Rectangle

See also

setShape(), setPolygon(), shape()

Definition at line 216 of file qwt_plot_shapeitem.cpp.

14.98.5.15 setRenderTolerance() `void QwtPlotShapeItem::setRenderTolerance ( double tolerance )`

Set the tolerance for the weeding optimization.

After translating the shape into target device coordinate (usually widget geometries) the painter path can be simplified by a point weeding algorithm (Douglas-Peucker).

For shapes built from curves and ellipses weeding might have the opposite effect because they have to be expanded to polygons.

Parameters

tolerance	Accepted error when reducing the number of points A value <= 0.0 disables weeding.

See also

renderTolerance(), QwtWeedingCurveFitter

Definition at line 357 of file qwt_plot_shapeitem.cpp.

14.98.5.16 setShape() `void QwtPlotShapeItem::setShape ( const QPainterPath & shape )`

Set the shape to be displayed.

Parameters

shape	Shape

See also

setShape(), shape()

Definition at line 244 of file qwt_plot_shapeitem.cpp.

14.98.5.17 shape() `QPainterPath QwtPlotShapeItem::shape ( ) const`

Returns

Shape to be displayed

See also

setShape()

Definition at line 266 of file qwt_plot_shapeitem.cpp.

14.98.5.18 testPaintAttribute() `bool QwtPlotShapeItem::testPaintAttribute ( PaintAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

setPaintAttribute()

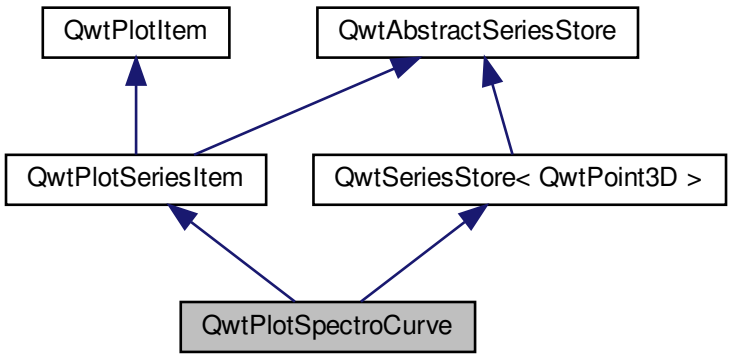
Definition at line 175 of file qwt_plot_shapeitem.cpp.

14.99 QwtPlotSpectroCurve Class Reference

Curve that displays 3D points as dots, where the z coordinate is mapped to a color.

`#include <qwt_plot_spectrocurve.h>`

Inheritance diagram for QwtPlotSpectroCurve:



Public Types

- enum PaintAttribute { ClipPoints = 1 }

  Paint attributes.
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPlotSpectroCurve (const QString &title=QString())
- QwtPlotSpectroCurve (const QwtText &title)
- virtual ~QwtPlotSpectroCurve ()

  Destructor.
- virtual int rtti () const override
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setSamples (const QVector< QwtPoint3D > &)
- void setSamples (QwtSeriesData< QwtPoint3D > *)
- void setColorMap (QwtColorMap *)
- const QwtColorMap * colorMap () const
- void setColorRange (const QwtInterval &)
- QwtInterval & colorRange () const
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- void setPenWidth (double)
- double penWidth () const

Protected Member Functions

- virtual void drawDots (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const

14.99.1 Detailed Description

Curve that displays 3D points as dots, where the z coordinate is mapped to a color.

Definition at line 22 of file qwt_plot_spectrocurve.h.

14.99.2 Member Typedef Documentation

14.99.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotSpectroCurve::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 34 of file qwt_plot_spectrocurve.h.

14.99.3 Member Enumeration Documentation

14.99.3.1 PaintAttribute `enum QwtPlotSpectroCurve::PaintAttribute`

Paint attributes.

Enumerator

ClipPoints	Clip points outside the canvas rectangle.

Definition at line 28 of file qwt_plot_spectrocurve.h.

14.99.4 Constructor & Destructor Documentation

14.99.4.1 QwtPlotSpectroCurve() [1/2]

```
QwtPlotSpectroCurve::QwtPlotSpectroCurve (
    const QString & title = QString() ) [explicit]
```

Constructor

Parameters

title	Title of the curve

Definition at line 55 of file qwt_plot_spectrocurve.cpp.

14.99.4.2 QwtPlotSpectroCurve() [2/2]

```
QwtPlotSpectroCurve::QwtPlotSpectroCurve (
    const QwtText & title ) [explicit]
```

Constructor

Parameters

title	Title of the curve

Definition at line 45 of file qwt_plot_spectrocurve.cpp.

14.99.5 Member Function Documentation

14.99.5.1 colorMap()

```
const QwtColorMap * QwtPlotSpectroCurve::colorMap ( ) const
```

Returns

Color Map used for mapping the intensity values to colors

See also

setColorMap(), setColorRange(), QwtColorMap::color()

Definition at line 163 of file qwt_plot_spectrocurve.cpp.

14.99.5.2 colorRange() `QwtInterval & QwtPlotSpectroCurve::colorRange ( ) const`

Returns

Value interval, that corresponds to the color map

See also

setColorRange(), setColorMap(), QwtColorMap::color()

Definition at line 191 of file qwt_plot_spectrocurve.cpp.

14.99.5.3 drawDots()

```
void QwtPlotSpectroCurve::drawDots (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw a subset of the points

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the series will be painted to its last sample.

See also

drawSeries()

Definition at line 270 of file qwt_plot_spectrocurve.cpp.

14.99.5.4 drawSeries()

```
void QwtPlotSpectroCurve::drawSeries (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [override], [virtual]
```

Draw a subset of the points

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to $<$ 0 the series will be painted to its last sample.

See also

drawDots()

Implements QwtPlotSeriesItem.

Definition at line 238 of file qwt_plot_spectrocurve.cpp.

14.99.5.5 penWidth() `double QwtPlotSpectroCurve::penWidth ( ) const`

Returns

Pen width used to draw a dot

See also

setPenWidth()

Definition at line 220 of file qwt_plot_spectrocurve.cpp.

14.99.5.6 rtti() `int QwtPlotSpectroCurve::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotSpectroCurve

Reimplemented from QwtPlotItem.

Definition at line 82 of file qwt_plot_spectrocurve.cpp.

14.99.5.7 setColorMap() `void QwtPlotSpectroCurve::setColorMap ( QwtColorMap * colorMap )`

Change the color map

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

Parameters

colorMap	Color Map

See also

colorMap(), setColorRange(), QwtColorMap::color(), QwtScaleWidget::setColorBarEnabled(), QwtScaleWidget::setColorMap()

Definition at line 147 of file qwt_plot_spectrocurve.cpp.

14.99.5.8 setColorRange()

```
void QwtPlotSpectroCurve::setColorRange (
    const QwtInterval & interval )
```

Set the value interval, that corresponds to the color map

Parameters

interval	interval.minValue() corresponds to 0.0, interval.maxValue() to 1.0 on the color map.

See also

colorRange(), setColorMap(), QwtColorMap::color()

Definition at line 176 of file qwt_plot_spectrocurve.cpp.

14.99.5.9 setPaintAttribute()

```
void QwtPlotSpectroCurve::setPaintAttribute (
    PaintAttribute attribute,
    bool on = true )
```

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off /sa PaintAttribute, testPaintAttribute()

Definition at line 94 of file qwt_plot_spectrocurve.cpp.

14.99.5.10 setPenWidth()

```
void QwtPlotSpectroCurve::setPenWidth (
    double penWidth )
```

Assign a pen width

Parameters

penWidth	New pen width

See also

penWidth()

Definition at line 202 of file qwt_plot_spectrocurve.cpp.

14.99.5.11 setSamples() [1/2]

```
void QwtPlotSpectroCurve::setSamples (
    const QVector< QwtPoint3D > & samples )
```

Initialize data with an array of samples.

Parameters

samples	Vector of points

Definition at line 115 of file qwt_plot_spectrocurve.cpp.

14.99.5.12 setSamples() [2/2]

```
void QwtPlotSpectroCurve::setSamples (
    QwtSeriesData< QwtPoint3D > * data )
```

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 130 of file qwt_plot_spectrocurve.cpp.

14.99.5.13 testPaintAttribute()

```
bool QwtPlotSpectroCurve::testPaintAttribute (
    PaintAttribute attribute ) const
```

Returns

True, when attribute is enabled

See also

PaintAttribute, setPaintAttribute()

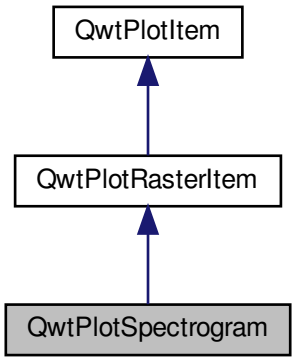
Definition at line 106 of file qwt_plot_spectrocurve.cpp.

14.100 QwtPlotSpectrogram Class Reference

A plot item, which displays a spectrogram.

`#include <qwt_plot_spectrogram.h>`

Inheritance diagram for QwtPlotSpectrogram:



Public Types

- enum DisplayMode { ImageMode = 0x01 , ContourMode = 0x02 }
- typedef QFlags< DisplayMode > DisplayModes

Public Member Functions

- QwtPlotSpectrogram (const QString &title=QString())
- virtual ~QwtPlotSpectrogram ()

  Destructor.
- void setDisplayMode (DisplayMode, bool on=true)
- bool testDisplayMode (DisplayMode) const
- void setData (QwtRasterData *data)
- const QwtRasterData * data () const
- QwtRasterData * data ()

- void setColorMap (QwtColorMap ∗)
- const QwtColorMap ∗ colorMap () const
- void setColorTableSize (int numColors)
- int colorTableSize () const
- virtual QwtInterval interval (Qt::Axis) const override
- virtual QRectF pixelHint (const QRectF &) const override

  Pixel hint.
- void setDefaultContourPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setDefaultContourPen (const QPen &)

  Set the default pen for the contour lines.
- QPen defaultContourPen () const
- virtual QPen contourPen (double level) const

  Calculate the pen for a contour line.
- void setConrecFlag (QwtRasterData::ConrecFlag, bool on)
- bool testConrecFlag (QwtRasterData::ConrecFlag) const
- void setContourLevels (const QList< double > &)
- QList< double > contourLevels () const
- virtual int rtti () const override
- virtual void draw (QPainter ∗, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const override

  Draw the spectrogram.

Protected Member Functions

- virtual QImage renderImage (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &area, const QSize &imageSize) const override

  Render an image from data and color map.
- virtual QSize contourRasterSize (const QRectF &, const QRect &) const

  Return the raster to be used by the CONREC contour algorithm.
- virtual QwtRasterData::ContourLines renderContourLines (const QRectF &rect, const QSize &raster) const
- virtual void drawContourLines (QPainter ∗, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtRasterData::ContourLines &) const
- void renderTile (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &tile, QImage ∗) const

  Render a tile of an image.

14.100.1 Detailed Description

A plot item, which displays a spectrogram.

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

On multi-core systems the performance of the image composition can often be improved by dividing the area into tiles - each of them rendered in a different thread (see QwtPlotItem::setRenderThreadCount()).

In ContourMode contour lines are painted for the contour levels.

See also

QwtRasterData, QwtColorMap, QwtPlotItem::setRenderThreadCount()

Definition at line 36 of file qwt_plot_spectrogram.h.

14.100.2 Member Typedef Documentation

14.100.2.1 DisplayModes `typedef QFlags<DisplayMode > QwtPlotSpectrogram::DisplayModes`

An ORed combination of DisplayMode values.

Definition at line 53 of file qwt_plot_spectrogram.h.

14.100.3 Member Enumeration Documentation

14.100.3.1 DisplayMode `enum QwtPlotSpectrogram::DisplayMode`

The display mode controls how the raster data will be represented.

See also

setDisplayMode(), testDisplayMode()

Enumerator

ImageMode	The values are mapped to colors using a color map.
ContourMode	The data is displayed using contour lines.

Definition at line 44 of file qwt_plot_spectrogram.h.

14.100.4 Constructor & Destructor Documentation

14.100.4.1 QwtPlotSpectrogram() `QwtPlotSpectrogram::QwtPlotSpectrogram ( const QString & title = QString() ) [explicit]`

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

The z value is initialized by 8.0.

Parameters

title	Title

See also

QwtPlotItem::setItemAttribute(), QwtPlotItem::setZ()

Definition at line 108 of file qwt_plot_spectrogram.cpp.

14.100.5 Member Function Documentation

14.100.5.1 colorMap() `const QwtColorMap * QwtPlotSpectrogram::colorMap ( ) const`

Returns

Color Map used for mapping the intensity values to colors

See also

setColorMap()

Definition at line 200 of file qwt_plot_spectrogram.cpp.

14.100.5.2 colorTableSize() `int QwtPlotSpectrogram::colorTableSize ( ) const`

Returns

Size of the color table, 0 means not using a color table

See also

QwtColorMap::colorTable(), setColorTableSize()

Definition at line 238 of file qwt_plot_spectrogram.cpp.

14.100.5.3 contourLevels() `QList< double > QwtPlotSpectrogram::contourLevels ( ) const`

Returns

Levels of the contour lines.

The levels are sorted in increasing order.

See also

contourLevels(), renderContourLines(), QwtRasterData::contourLines()

Definition at line 382 of file qwt_plot_spectrogram.cpp.

14.100.5.4 contourPen() `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

level	Contour level

Returns

Pen for the contour line

Note

contourPen is only used if defaultContourPen().style() == Qt::NoPen

See also

setDefaultContourPen(), setColorMap(), setContourLevels()

Definition at line 303 of file qwt_plot_spectrogram.cpp.

14.100.5.5 contourRasterSize()

```
QSize QwtPlotSpectrogram::contourRasterSize (
    const QRectF & 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 the resolution depending on pixelSize().

Parameters

area	Rectangle, where to calculate the contour lines
rect	Rectangle 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()

Definition at line 663 of file qwt_plot_spectrogram.cpp.

14.100.5.6 data() [1/2] QwtRasterData * QwtPlotSpectrogram::data ( )

Returns

Spectrogram data

See also

setData()

Definition at line 418 of file qwt_plot_spectrogram.cpp.

14.100.5.7 data() [2/2] const QwtRasterData * QwtPlotSpectrogram::data ( ) const

Returns

Spectrogram data

See also

setData()

Definition at line 409 of file qwt_plot_spectrogram.cpp.

14.100.5.8 defaultContourPen() QPen QwtPlotSpectrogram::defaultContourPen ( ) const

Returns

Default contour pen

See also

setDefaultContourPen()

Definition at line 287 of file qwt_plot_spectrogram.cpp.

14.100.5.9 draw()

```
void QwtPlotSpectrogram::draw (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const  [override], [virtual]
```

Draw the spectrogram.

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas in painter coordinates

See also

setDisplayMode(), renderImage(), QwtPlotRasterItem::draw(), drawContourLines()

Reimplemented from QwtPlotRasterItem.

Definition at line 754 of file qwt_plot_spectrogram.cpp.

14.100.5.10 drawContourLines()

```
void QwtPlotSpectrogram::drawContourLines (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QwtRasterData::ContourLines & contourLines ) const  [protected], [virtual]
```

Paint the contour lines

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
contourLines	Contour lines

See also

renderContourLines(), defaultContourPen(), contourPen()

Definition at line 709 of file qwt_plot_spectrogram.cpp.

14.100.5.11 interval()

```
QwtInterval QwtPlotSpectrogram::interval (
    Qt::Axis axis ) const  [override], [virtual]
```

Returns

Bounding interval for an axis

The default implementation returns the interval of the associated raster data object.

Parameters

axis	X, Y, or Z axis

See also

QwtRasterData::interval()

Reimplemented from QwtPlotRasterItem.

Definition at line 432 of file qwt_plot_spectrogram.cpp.

14.100.5.12 pixelHint()

```
QRectF QwtPlotSpectrogram::pixelHint (
        const QRectF & area ) const  [override], [virtual]
```

Pixel hint.

The geometry of a pixel is used to calculated the resolution and alignment of the rendered image.

The default implementation returns data()->pixelHint(rect);

Parameters

area	In most implementations the resolution of the data doesn't depend on the requested area.

Returns

Bounding rectangle of a pixel

See also

QwtPlotRasterItem::pixelHint(), QwtRasterData::pixelHint(), render(), renderImage()

Reimplemented from QwtPlotRasterItem.

Definition at line 456 of file qwt_plot_spectrogram.cpp.

14.100.5.13 renderContourLines()

```
QwtRasterData::ContourLines QwtPlotSpectrogram::renderContour↩
Lines (
        const QRectF & rect,
        const QSize & raster ) const  [protected], [virtual]
```

Calculate contour lines

Parameters

rect	Rectangle, where to calculate the contour lines
raster	Raster, used by the CONREC algorithm

Returns

Calculated contour lines

See also

contourLevels(), setConrecFlag(), QwtRasterData::contourLines()

Definition at line 689 of file qwt_plot_spectrogram.cpp.

14.100.5.14 renderImage()

```
QImage QwtPlotSpectrogram::renderImage (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & area,
    const QSize & imageSize ) const  [override], [protected], [virtual]
```

Render an image from data and color map.

For each pixel of area the value is mapped into a color.

Parameters

xMap	X-Scale Map
yMap	Y-Scale Map
area	Requested area for the image in scale coordinates
imageSize	Size of the requested image

Returns

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

See also

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

Implements QwtPlotRasterItem.

Definition at line 480 of file qwt_plot_spectrogram.cpp.

14.100.5.15 renderTile()

```
void QwtPlotSpectrogram::renderTile (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRect & tile,
    QImage * image ) const  [protected]
```

Render a tile of an image.

Rendering in tiles can be used to composite an image in parallel threads.

Parameters

xMap	X-Scale Map
yMap	Y-Scale Map
tile	Geometry of the tile in image coordinates
image	Image to be rendered

Definition at line 572 of file qwt_plot_spectrogram.cpp.

14.100.5.16 rtti()

```
int QwtPlotSpectrogram::rtti ( ) const  [override], [virtual]
```

Returns

QwtPlotItem::Rtti_PlotSpectrogram

Reimplemented from QwtPlotItem.

Definition at line 126 of file qwt_plot_spectrogram.cpp.

14.100.5.17 setColorMap()

```
void QwtPlotSpectrogram::setColorMap (
    QwtColorMap * colorMap )
```

Change the color map

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

Parameters

colorMap	Color Map

See also

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

Definition at line 177 of file qwt_plot_spectrogram.cpp.

14.100.5.18 setColorTableSize() void QwtPlotSpectrogram::setColorTableSize (int *numColors*)

Limit the number of colors being used by the color map

When using a color table the mapping from the value into a color is usually faster as it can be done by simple lookups into a precalculated color table.

Setting a table size $>$ 0 enables using a color table, while setting the size to 0 disables it.

The default size = 0, and no color table is used.

Parameters

numColors	Number of colors. 0 means not using a color table

Note

The colorTableSize has no effect when using a color table of QwtColorMap::Indexed, where the size is always 256.

See also

QwtColorMap::colorTable(), colorTableSize()

Definition at line 224 of file qwt_plot_spectrogram.cpp.

14.100.5.19 setConrecFlag() void QwtPlotSpectrogram::setConrecFlag (QwtRasterData::ConrecFlag *flag*, bool *on*)

Modify an attribute of the CONREC algorithm, used to calculate the contour lines.

Parameters

flag	CONREC flag
on	On/Off

See also

testConrecFlag(), renderContourLines(), QwtRasterData::contourLines()

Definition at line 324 of file qwt_plot_spectrogram.cpp.

14.100.5.20 setContourLevels() void QwtPlotSpectrogram::setContourLevels (const QList< double > & *levels*)

Set the levels of the contour lines

Parameters

levels	Values of the contour levels

See also

contourLevels(), renderContourLines(), QwtRasterData::contourLines()

Note

contourLevels returns the same levels but sorted.

Definition at line 365 of file qwt_plot_spectrogram.cpp.

14.100.5.21 setData()

```
void QwtPlotSpectrogram::setData (
    QwtRasterData * data )
```

Set the data to be displayed

Parameters

data	Spectrogram Data

See also

data()

Definition at line 393 of file qwt_plot_spectrogram.cpp.

14.100.5.22 setDefaultContourPen() [1/2]

```
void QwtPlotSpectrogram::setDefaultContourPen (
    const QColor & color,
    qreal width = 0.0,
    Qt::PenStyle style = Qt::SolidLine )
```

Build and assign the default pen for the contour lines

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 256 of file qwt_plot_spectrogram.cpp.

14.100.5.23 setDefaultContourPen() [2/2]

```
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()

Definition at line 272 of file qwt_plot_spectrogram.cpp.

14.100.5.24 setDisplayMode()

```
void QwtPlotSpectrogram::setDisplayMode (
    DisplayMode mode,
    bool on = true )
```

The display mode controls how the raster data will be represented.

Parameters

mode	Display mode
on	On/Off

The default setting enables ImageMode.

See also

DisplayMode, displayMode()

Definition at line 141 of file qwt_plot_spectrogram.cpp.

14.100.5.25 testConrecFlag()

```
bool QwtPlotSpectrogram::testConrecFlag (
    QwtRasterData::ConrecFlag flag ) const
```

Test an attribute of the CONREC algorithm, used to calculate the contour lines.

Parameters

flag	CONREC flag

Returns

true, is enabled

The default setting enables QwtRasterData::IgnoreAllVerticesOnLevel

See also

setConrecClag(), renderContourLines(), QwtRasterData::contourLines()

Definition at line 350 of file qwt_plot_spectrogram.cpp.

14.100.5.26 testDisplayMode() `bool QwtPlotSpectrogram::testDisplayMode ( DisplayMode mode ) const`

The display mode controls how the raster data will be represented.

Parameters

mode	Display mode

Returns

true if mode is enabled

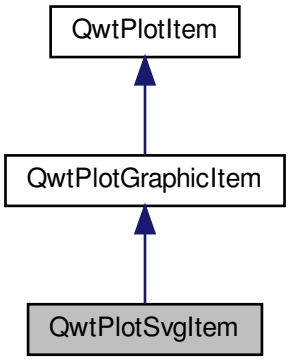
Definition at line 161 of file qwt_plot_spectrogram.cpp.

14.101 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 Member Functions

- QwtPlotSvgItem (const QString &title=QString())

  Constructor.
- QwtPlotSvgItem (const QwtText &title)

  Constructor.
- virtual ~QwtPlotSvgItem ()

  Destructor.
- bool loadFile (const QRectF &, const QString &fileName)
- bool loadData (const QRectF &, const QByteArray &)

Additional Inherited Members

14.101.1 Detailed Description

A plot item, which displays data in Scalable Vector Graphics (SVG) format.

SVG images are often used to display maps

QwtPlotSvgItem is only a small convenience wrapper class for QwtPlotGraphicItem, that creates a QwtGraphic from SVG data.

Definition at line 28 of file qwt_plot_svgitem.h.

14.101.2 Constructor & Destructor Documentation

14.101.2.1 QwtPlotSvgItem() [1/2]

```
QwtPlotSvgItem::QwtPlotSvgItem (
        const QString & title = QString() ) [explicit]
```

Constructor.

Parameters

title	Title

Definition at line 20 of file qwt_plot_svgitem.cpp.

14.101.2.2 QwtPlotSvgItem() [2/2] QwtPlotSvgItem::QwtPlotSvgItem (const QwtText & *title*) [explicit]

Constructor.

Parameters

title	Title

Definition at line 29 of file qwt_plot_svgitem.cpp.

14.101.3 Member Function Documentation

14.101.3.1 loadData() bool QwtPlotSvgItem::loadData (const QRectF & *rect*, const QByteArray & *data*)

Load SVG data

Parameters

rect	Bounding rectangle
data	in SVG format

Returns

true, if the SVG data could be loaded

Definition at line 74 of file qwt_plot_svgitem.cpp.

14.101.3.2 loadFile() bool QwtPlotSvgItem::loadFile (const QRectF & *rect*, const QString & *fileName*)

Load a SVG file

Parameters

rect	Bounding rectangle
fileName	SVG file name

Returns

true, if the SVG file could be loaded

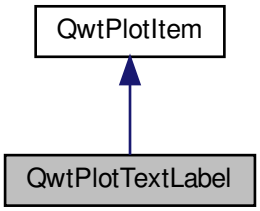
Definition at line 47 of file qwt_plot_svgitem.cpp.

14.102 QwtPlotTextLabel Class Reference

A plot item, which displays a text label.

`#include <qwt_plot_textlabel.h>`

Inheritance diagram for QwtPlotTextLabel:



Public Member Functions

- QwtPlotTextLabel ()

  Constructor.
- virtual ~QwtPlotTextLabel ()

  Destructor.
- virtual int rtti () const override
- void setText (const QwtText &)
- QwtText text () const
- void setMargin (int margin)
- int margin () const
- virtual QRectF textRect (const QRectF &, const QSizeF &) const

  Align the text label.

Protected Member Functions

- virtual void draw (QPainter *, const QwtScaleMap &, const QwtScaleMap &, const QRectF &) const override
- void invalidateCache ()

  Invalidate all internal cache.

Additional Inherited Members

14.102.1 Detailed Description

A plot item, which displays a text label.

QwtPlotTextLabel displays a text label aligned to the plot canvas.

In opposite to QwtPlotMarker the position of the label is unrelated to plot coordinates.

As drawing a text is an expensive operation the label is cached in a pixmap to speed up replots.

Example

The following code shows how to add a title.

```
QwtText title( "Plot Title" );
title.setRenderFlags( Qt::AlignHCenter | Qt::AlignTop );
QFont font;
font.setBold( true );
title.setFont( font );
QwtPlotTextLabel *titleItem = new QwtPlotTextLabel();
titleItem->setText( title );
titleItem->attach( plot );
```

See also

QwtPlotMarker

Definition at line 47 of file qwt_plot_textlabel.h.

14.102.2 Constructor & Destructor Documentation

14.102.2.1 QwtPlotTextLabel() `QwtPlotTextLabel::QwtPlotTextLabel ( )`

Constructor.

Initializes an text label with an empty text

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

The z value is initialized by 150

See also

QwtPlotItem::setItemAttribute(), QwtPlotItem::setZ()

Definition at line 82 of file qwt_plot_textlabel.cpp.

14.102.3 Member Function Documentation

14.102.3.1 draw() `void QwtPlotTextLabel::draw ( QPainter * painter, const QwtScaleMap & xMap, const QwtScaleMap & yMap, const QRectF & canvasRect ) const [override], [protected], [virtual]`

Draw the text label

Parameters

painter	Painter
xMap	x Scale Map
yMap	y Scale Map
canvasRect	Contents rectangle of the canvas in painter coordinates

See also

textRect()

Implements QwtPlotItem.

Definition at line 176 of file qwt_plot_textlabel.cpp.

14.102.3.2 margin() `int QwtPlotTextLabel::margin ( ) const`

Returns

Margin added to the contentsMargins() of the canvas

See also

setMargin()

Definition at line 160 of file qwt_plot_textlabel.cpp.

14.102.3.3 rtti() `int QwtPlotTextLabel::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotTextLabel

Reimplemented from QwtPlotItem.

Definition at line 100 of file qwt_plot_textlabel.cpp.

14.102.3.4 setMargin() `void QwtPlotTextLabel::setMargin ( int margin )`

Set the margin

The margin is the distance between the contentsRect() of the plot canvas and the rectangle where the label can be displayed.

Parameters

margin	Margin

See also

margin(), textRect()

Definition at line 146 of file qwt_plot_textlabel.cpp.

14.102.3.5 setText() `void QwtPlotTextLabel::setText ( const QwtText & text )`

Set the text

The label will be aligned to the plot canvas according to the alignment flags of text.

Parameters

text	Text to be displayed

See also

text(), QwtText::renderFlags()

Definition at line 115 of file qwt_plot_textlabel.cpp.

14.102.3.6 text() `QwtText QwtPlotTextLabel::text ( ) const`

Returns

Text to be displayed

See also

setText()

Definition at line 130 of file qwt_plot_textlabel.cpp.

14.102.3.7 textRect() `QRectF QwtPlotTextLabel::textRect ( const QRectF & rect, const QSizeF & textSize ) const [virtual]`

Align the text label.

Parameters

rect	Canvas rectangle with margins subtracted
textSize	Size required to draw the text

Returns

A rectangle aligned according the the alignment flags of the text.

See also

setMargin(), QwtText::renderFlags(), QwtText::textSize()

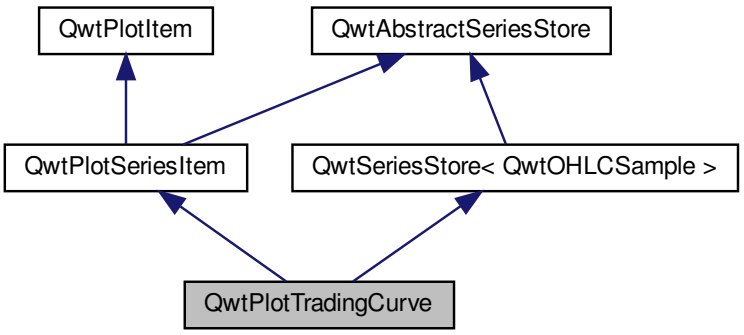
Definition at line 263 of file qwt_plot_textlabel.cpp.

14.103 QwtPlotTradingCurve Class Reference

QwtPlotTradingCurve illustrates movements in the price of a financial instrument over time.

`#include <qwt_plot_tradingcurve.h>`

Inheritance diagram for QwtPlotTradingCurve:



Public Types

- enum SymbolStyle { NoSymbol = -1 , Bar , CandleStick , UserSymbol = 100 }
  Symbol styles.
- enum Direction { Increasing , Decreasing }
  Direction of a price movement.
- enum PaintAttribute { ClipSymbols = 0x01 }
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPlotTradingCurve (const QString &title=QString())
- QwtPlotTradingCurve (const QwtText &title)
- virtual ~QwtPlotTradingCurve ()

  Destructor.
- virtual int rtti () const override
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setSamples (const QVector< QwtOHLCSample > &)
- void setSamples (QwtSeriesData< QwtOHLCSample > *)
- void setSymbolStyle (SymbolStyle style)
- SymbolStyle symbolStyle () const
- void setSymbolPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setSymbolPen (const QPen &)

  Set the symbol pen.
- QPen symbolPen () const
- void setSymbolBrush (Direction, const QBrush &)
- QBrush symbolBrush (Direction) const
- void setSymbolExtent (double)

  Set the extent of the symbol.
- double symbolExtent () const
- void setMinSymbolWidth (double)
- double minSymbolWidth () const
- void setMaxSymbolWidth (double)
- double maxSymbolWidth () const
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual QRectF boundingRect () const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override

Protected Member Functions

- void init ()

  Initialize internal members.
- virtual void drawSymbols (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void drawUserSymbol (QPainter *, SymbolStyle, const QwtOHLCSample &, Qt::Orientation, bool inverted, double symbolWidth) const

  Draw a symbol for a symbol style >= UserSymbol.
- void drawBar (QPainter *, const QwtOHLCSample &, Qt::Orientation, bool inverted, double width) const

  Draw a bar.
- void drawCandleStick (QPainter *, const QwtOHLCSample &, Qt::Orientation, double width) const

  Draw a candle stick.
- virtual double scaledSymbolWidth (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect) const

14.103.1 Detailed Description

QwtPlotTradingCurve illustrates movements in the price of a financial instrument over time.

QwtPlotTradingCurve supports candlestick or bar (OHLC) charts that are used in the domain of technical analysis.

While the length (height or width depending on orientation()) of each symbol depends on the corresponding OHLC sample the size of the other dimension can be controlled using:

- setSymbolExtent()
- setSymbolMinWidth()
- setSymbolMaxWidth()

The extent is a size in scale coordinates, so that the symbol width is increasing when the plot is zoomed in. Minimum/Maximum width is in widget coordinates independent from the zoom level. When setting the minimum and maximum to the same value, the width of the symbol is fixed.

Definition at line 37 of file qwt_plot_tradingcurve.h.

14.103.2 Member Typedef Documentation

14.103.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPlotTradingCurve::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 101 of file qwt_plot_tradingcurve.h.

14.103.3 Member Enumeration Documentation

14.103.3.1 Direction `enum QwtPlotTradingCurve::Direction`

Direction of a price movement.

Enumerator

Increasing	The closing price is higher than the opening price.
Decreasing	The closing price is lower than the opening price.

Definition at line 82 of file qwt_plot_tradingcurve.h.

14.103.3.2 PaintAttribute enum QwtPlotTradingCurve::PaintAttribute

Attributes to modify the drawing algorithm.

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

ClipSymbols	Check if a symbol is on the plot canvas before painting it.

Definition at line 95 of file qwt_plot_tradingcurve.h.

14.103.3.3 SymbolStyle enum QwtPlotTradingCurve::SymbolStyle

Symbol styles.

The default setting is QwtPlotSeriesItem::CandleStick.

See also

setSymbolStyle(), symbolStyle()

Enumerator

NoSymbol	Nothing is displayed.
Bar	A line on the chart shows the price range (the highest and lowest prices) over one unit of time, e.g. one day or one hour. Tick marks project from each side of the line indicating the opening and closing price.
CandleStick	The range between opening/closing price are displayed as a filled box. The fill brush depends on the direction of the price movement. The box is connected to the highest/lowest values by lines.
UserSymbol	SymbolTypes >= UserSymbol are displayed by drawUserSymbol(), that needs to be overloaded and implemented in derived curve classes. **See also** drawUserSymbol()

Definition at line 48 of file qwt_plot_tradingcurve.h.

14.103.4 Constructor & Destructor Documentation

14.103.4.1 QwtPlotTradingCurve() [1/2] QwtPlotTradingCurve::QwtPlotTradingCurve (const QString & *title* = *QString()*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 70 of file qwt_plot_tradingcurve.cpp.

14.103.4.2 QwtPlotTradingCurve() [2/2] QwtPlotTradingCurve::QwtPlotTradingCurve (const QwtText & *title*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 60 of file qwt_plot_tradingcurve.cpp.

14.103.5 Member Function Documentation

14.103.5.1 boundingRect() QRectF QwtPlotTradingCurve::boundingRect ( ) const [override], [virtual]

Returns

Bounding rectangle of all samples. For an empty series the rectangle is invalid.

Reimplemented from QwtPlotSeriesItem.

Definition at line 365 of file qwt_plot_tradingcurve.cpp.

14.103.5.2 drawBar() void QwtPlotTradingCurve::drawBar (QPainter * *painter*, const QwtOHLCSample & *sample*, Qt::Orientation *orientation*, bool *inverted*, double *width*) const [protected]

Draw a bar.

Parameters

painter	Qt painter, initialized with pen/brush
sample	Sample, already translated into paint device coordinates
orientation	Vertical or horizontal
inverted	When inverted is false the open tick is painted to the left/top, otherwise it is painted right/bottom. The close tick is painted in the opposite direction of the open tick. painted in the opposite d opposite direction.
width	Width or height of the candle, depending on the orientation

See also

Bar

Definition at line 562 of file qwt_plot_tradingcurve.cpp.

14.103.5.3 drawCandleStick()

```
void QwtPlotTradingCurve::drawCandleStick (
    QPainter * painter,
    const QwtOHLCSample & sample,
    Qt::Orientation orientation,
    double width ) const  [protected]
```

Draw a candle stick.

Parameters

painter	Qt painter, initialized with pen/brush
sample	Samples already translated into paint device coordinates
orientation	Vertical or horizontal
width	Width or height of the candle, depending on the orientation

See also

CandleStick

Definition at line 601 of file qwt_plot_tradingcurve.cpp.

14.103.5.4 drawSeries()

```
void QwtPlotTradingCurve::drawSeries (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [override], [virtual]
```

Draw an interval of the curve

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

See also

drawSymbols()

Implements QwtPlotSeriesItem.

Definition at line 387 of file qwt_plot_tradingcurve.cpp.

14.103.5.5 drawSymbols()

```
void QwtPlotTradingCurve::drawSymbols (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw symbols

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	Index of the first point to be painted
to	Index of the last point to be painted

See also

drawSeries()

Definition at line 420 of file qwt_plot_tradingcurve.cpp.

14.103.5.6 drawUserSymbol()

```
void QwtPlotTradingCurve::drawUserSymbol (
    QPainter * painter,
    SymbolStyle symbolStyle,
    const QwtOHLCSample & sample,
    Qt::Orientation orientation,
    bool inverted,
    double symbolWidth ) const  [protected], [virtual]
```

Draw a symbol for a symbol style >= UserSymbol.

The implementation does nothing and is intended to be overloaded

Parameters

painter	Qt painter, initialized with pen/brush

Parameters

symbolStyle	Symbol style
sample	Samples already translated into paint device coordinates
orientation	Vertical or horizontal
inverted	True, when the opposite scale (Qt::Vertical: x, Qt::Horizontal: y) is increasing in the opposite direction as QPainter coordinates.
symbolWidth	Width of the symbol in paint device coordinates

Definition at line 534 of file qwt_plot_tradingcurve.cpp.

14.103.5.7 legendIcon()

```
QwtGraphic QwtPlotTradingCurve::legendIcon (
    int index,
    const QSizeF & size ) const  [override], [virtual]
```

Returns

A rectangle filled with the color of the symbol pen

Parameters

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

See also

setLegendIconSize(), legendData()

Reimplemented from QwtPlotItem.

Definition at line 642 of file qwt_plot_tradingcurve.cpp.

14.103.5.8 maxSymbolWidth()

```
double QwtPlotTradingCurve::maxSymbolWidth ( ) const
```

Returns

Maximum for the symbol width

See also

setMaxSymbolWidth(), minSymbolWidth(), symbolExtent()

Definition at line 356 of file qwt_plot_tradingcurve.cpp.

14.103.5.9 minSymbolWidth() `double QwtPlotTradingCurve::minSymbolWidth ( ) const`

Returns

Minmum for the symbol width

See also

setMinSymbolWidth(), maxSymbolWidth(), symbolExtent()

Definition at line 328 of file qwt_plot_tradingcurve.cpp.

14.103.5.10 rtti() `int QwtPlotTradingCurve::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotTradingCurve

Reimplemented from QwtPlotItem.

Definition at line 95 of file qwt_plot_tradingcurve.cpp.

14.103.5.11 scaledSymbolWidth()

```
double QwtPlotTradingCurve::scaledSymbolWidth (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const [protected], [virtual]
```

Calculate the symbol width in paint coordinates

The width is calculated by scaling the symbol extent into paint device coordinates bounded by the minimum/maximum symbol width.

Parameters

xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas

Returns

Symbol width in paint coordinates

See also

symbolExtent(), minSymbolWidth(), maxSymbolWidth()

Definition at line 664 of file qwt_plot_tradingcurve.cpp.

14.103.5.12 setMaxSymbolWidth() `void QwtPlotTradingCurve::setMaxSymbolWidth ( double width )`

Set a maximum for the symbol width

A value <= 0.0 means an unlimited width

Parameters

width	Width in paint device coordinates

See also

maxSymbolWidth(), setMinSymbolWidth(), setSymbolExtent()

Definition at line 341 of file qwt_plot_tradingcurve.cpp.

14.103.5.13 setMinSymbolWidth() `void QwtPlotTradingCurve::setMinSymbolWidth ( double width )`

Set a minimum for the symbol width

Parameters

width	Width in paint device coordinates

See also

minSymbolWidth(), setMaxSymbolWidth(), setSymbolExtent()

Definition at line 312 of file qwt_plot_tradingcurve.cpp.

14.103.5.14 setPaintAttribute() `void QwtPlotTradingCurve::setPaintAttribute ( PaintAttribute attribute, bool on = true )`

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 107 of file qwt_plot_tradingcurve.cpp.

14.103.5.15 setSamples() [1/2] void QwtPlotTradingCurve::setSamples (const QVector< QwtOHLCSample > & *samples*)

Initialize data with an array of samples.

Parameters

samples	Vector of samples

See also

QwtPlotSeriesItem::setData()

Definition at line 132 of file qwt_plot_tradingcurve.cpp.

14.103.5.16 setSamples() [2/2] void QwtPlotTradingCurve::setSamples (QwtSeriesData< QwtOHLCSample > * *data*)

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 148 of file qwt_plot_tradingcurve.cpp.

14.103.5.17 setSymbolBrush() void QwtPlotTradingCurve::setSymbolBrush (Direction *direction*, const QBrush & *brush*)

Set the symbol brush

Parameters

direction	Direction type
brush	Brush used to fill the body of all candlestick symbols with the direction

See also

symbolBrush(), setSymbolPen()

Definition at line 238 of file qwt_plot_tradingcurve.cpp.

14.103.5.18 setSymbolExtent()

```
void QwtPlotTradingCurve::setSymbolExtent (
        double extent )
```

Set the extent of the symbol.

The width of the symbol is given in scale coordinates. When painting a symbol the width is scaled into paint device coordinates by scaledSymbolWidth(). The scaled width is bounded by minSymbolWidth(), maxSymbolWidth()

Parameters

extent	Symbol width in scale coordinates

See also

symbolExtent(), scaledSymbolWidth(), setMinSymbolWidth(), setMaxSymbolWidth()

Definition at line 284 of file qwt_plot_tradingcurve.cpp.

14.103.5.19 setSymbolPen() [1/2]

```
void QwtPlotTradingCurve::setSymbolPen (
        const QColor & color,
        qreal width = 0.0,
        Qt::PenStyle style = Qt::SolidLine )
```

Build and assign the symbol pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 195 of file qwt_plot_tradingcurve.cpp.

14.103.5.20 setSymbolPen() [2/2] void QwtPlotTradingCurve::setSymbolPen (const QPen & *pen*)

Set the symbol pen.

The symbol pen is used for rendering the lines of the bar or candlestick symbols

See also

symbolPen(), setSymbolBrush()

Definition at line 209 of file qwt_plot_tradingcurve.cpp.

14.103.5.21 setSymbolStyle() void QwtPlotTradingCurve::setSymbolStyle (SymbolStyle *style*)

Set the symbol style

Parameters

style	Symbol style

See also

symbolStyle(), setSymbolExtent(), setSymbolPen(), setSymbolBrush()

Definition at line 162 of file qwt_plot_tradingcurve.cpp.

14.103.5.22 symbolBrush() QBrush QwtPlotTradingCurve::symbolBrush (Direction *direction*) const

Parameters

direction	

Returns

Brush used to fill the body of all candlestick symbols with the direction

See also

setSymbolPen(), symbolBrush()

Definition at line 262 of file qwt_plot_tradingcurve.cpp.

14.103.5.23 symbolExtent() `double QwtPlotTradingCurve::symbolExtent ( ) const`

Returns

Extent of a symbol in scale coordinates

See also

setSymbolExtent(), scaledSymbolWidth(), minSymbolWidth(), maxSymbolWidth()

Definition at line 301 of file qwt_plot_tradingcurve.cpp.

14.103.5.24 symbolPen() `QPen QwtPlotTradingCurve::symbolPen ( ) const`

Returns

Symbol pen

See also

setSymbolPen(), symbolBrush()

Definition at line 224 of file qwt_plot_tradingcurve.cpp.

14.103.5.25 symbolStyle() `QwtPlotTradingCurve::SymbolStyle QwtPlotTradingCurve::symbolStyle ( ) const`

Returns

Symbol style

See also

setSymbolStyle(), symbolExtent(), symbolPen(), symbolBrush()

Definition at line 177 of file qwt_plot_tradingcurve.cpp.

14.103.5.26 testPaintAttribute() `bool QwtPlotTradingCurve::testPaintAttribute ( PaintAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

PaintAttribute, setPaintAttribute()

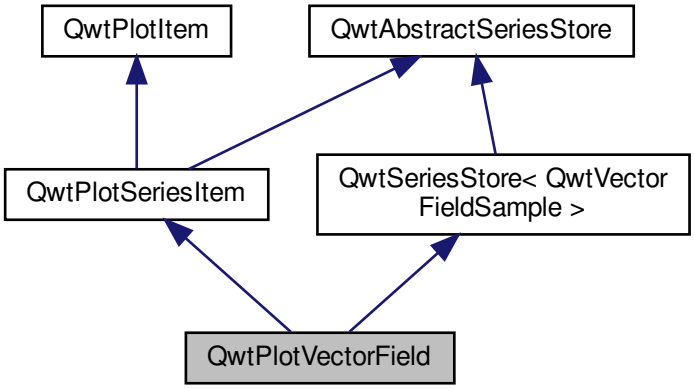
Definition at line 120 of file qwt_plot_tradingcurve.cpp.

14.104 QwtPlotVectorField Class Reference

A plot item, that represents a vector field.

`#include <qwt_plot_vectorfield.h>`

Inheritance diagram for QwtPlotVectorField:



Public Types

- enum IndicatorOrigin { OriginHead , OriginTail , OriginCenter }
- enum PaintAttribute { **FilterVectors** = 0x01 }
- enum MagnitudeMode { MagnitudeAsColor = 0x01 , MagnitudeAsLength = 0x02 }
- typedef QFlags< PaintAttribute > PaintAttributes
- typedef QFlags< MagnitudeMode > MagnitudeModes

Public Member Functions

- QwtPlotVectorField (const QString &title=QString())
- QwtPlotVectorField (const QwtText &title)
- virtual ~QwtPlotVectorField ()

  Destructor.
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setMagnitudeMode (MagnitudeMode, bool on=true)
- bool testMagnitudeMode (MagnitudeMode) const
- void setSymbol (QwtVectorFieldSymbol *)
- const QwtVectorFieldSymbol * symbol () const
- void setPen (const QPen &)
- QPen pen () const
- void setBrush (const QBrush &)

  Assign a brush.
- QBrush brush () const

- void setRasterSize (const QSizeF &)
- QSizeF rasterSize () const
- void setIndicatorOrigin (IndicatorOrigin)
- IndicatorOrigin indicatorOrigin () const
- void setSamples (const QVector< QwtVectorFieldSample > &)
- void setSamples (QwtVectorFieldData *)
- void setColorMap (QwtColorMap *)
- const QwtColorMap * colorMap () const
- void setMagnitudeRange (const QwtInterval &)
- QwtInterval magnitudeRange () const
- void setMinArrowLength (double)
- double minArrowLength () const
- void setMaxArrowLength (double)
- double maxArrowLength () const
- virtual double arrowLength (double magnitude) const
- virtual QRectF boundingRect () const override
- virtual void drawSeries (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const override
- virtual int rtti () const override
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const override
- void setMagnitudeScaleFactor (double factor)

  Set the magnitudeScaleFactor.
- double magnitudeScaleFactor () const

Protected Member Functions

- virtual void drawSymbols (QPainter *, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRectF &canvasRect, int from, int to) const
- virtual void drawSymbol (QPainter *, double x, double y, double vx, double vy) const
- virtual void dataChanged () override

  dataChanged() indicates, that the series has been changed.

14.104.1 Detailed Description

A plot item, that represents a vector field.

A vector field is a representation of a points with a given magnitude and direction as arrows. While the direction affects the direction of the arrow, the magnitude might be represented as a color or by the length of the arrow.

See also

QwtVectorFieldSymbol, QwtVectorFieldSample

Definition at line 30 of file qwt_plot_vectorfield.h.

14.104.2 Member Typedef Documentation

14.104.2.1 MagnitudeModes typedef QFlags<MagnitudeMode > QwtPlotVectorField::MagnitudeModes

An ORed combination of MagnitudeMode values.

Definition at line 90 of file qwt_plot_vectorfield.h.

14.104.2.2 PaintAttributes typedef QFlags<PaintAttribute > QwtPlotVectorField::PaintAttributes

An ORed combination of PaintAttribute values.

Definition at line 67 of file qwt_plot_vectorfield.h.

14.104.3 Member Enumeration Documentation

14.104.3.1 IndicatorOrigin enum QwtPlotVectorField::IndicatorOrigin

Depending on the origin the indicator symbol (usually an arrow) will be to the position of the corresponding sample.

Enumerator

OriginHead	symbol points to the sample position
OriginTail	The arrow starts at the sample position.
OriginCenter	The arrow is centered at the sample position.

Definition at line 39 of file qwt_plot_vectorfield.h.

14.104.3.2 MagnitudeMode enum QwtPlotVectorField::MagnitudeMode

Depending on the MagnitudeMode the magnitude component will have an impact on the attributes of the symbol/arrow.

See also

setMagnitudeMode()

Enumerator

MagnitudeAsColor	The magnitude will be mapped to a color using a color map **See also** magnitudeRange(), colorMap()
MagnitudeAsLength	The magnitude will have an impact on the length of the arrow/symbol **See also** arrowLength(), magnitudeScaleFactor()

Definition at line 75 of file qwt_plot_vectorfield.h.

14.104.3.3 PaintAttribute enum QwtPlotVectorField::PaintAttribute

Attributes to modify the rendering

See also

setPaintAttribute(), testPaintAttribute()

Definition at line 55 of file qwt_plot_vectorfield.h.

14.104.4 Constructor & Destructor Documentation

14.104.4.1 QwtPlotVectorField() [1/2] QwtPlotVectorField::QwtPlotVectorField (const QString & *title* = *QString()*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 302 of file qwt_plot_vectorfield.cpp.

14.104.4.2 QwtPlotVectorField() [2/2] QwtPlotVectorField::QwtPlotVectorField (const QwtText & *title*) [explicit]

Constructor

Parameters

title	Title of the curve

Definition at line 292 of file qwt_plot_vectorfield.cpp.

14.104.5 Member Function Documentation

14.104.5.1 arrowLength() `double QwtPlotVectorField::arrowLength ( double magnitude ) const [virtual]`

Computes length of the arrow in screen coordinate units based on its magnitude.

Default implementation simply scales the vector using the magnitudeScaleFactor() If the result is not null, the length is then bounded into the interval [minArrowLength(), maxArrowLength()].

Re-implement this function to provide special handling for zero/non-zero magnitude arrows, or impose minimum/maximum arrow length limits.

Parameters

magnitude	Magnitude

Returns

Length of arrow to be drawn in dependence of vector magnitude.

See also

magnitudeScaleFactor, minArrowLength(), maxArrowLength()

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 726 of file qwt_plot_vectorfield.cpp.

14.104.5.2 boundingRect() `QRectF QwtPlotVectorField::boundingRect ( ) const [override], [virtual]`

Returns

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

Note

A width or height < 0.0 is ignored by the autoscaler

Reimplemented from QwtPlotSeriesItem.

Definition at line 749 of file qwt_plot_vectorfield.cpp.

14.104.5.3 brush() `QBrush QwtPlotVectorField::brush ( ) const`

Returns

Brush used to fill the symbol

See also

setBrush(), pen()

Definition at line 379 of file qwt_plot_vectorfield.cpp.

14.104.5.4 colorMap() `const QwtColorMap * QwtPlotVectorField::colorMap ( ) const`

Returns

Color Map used for mapping the intensity values to colors

See also

setColorMap()

Definition at line 590 of file qwt_plot_vectorfield.cpp.

14.104.5.5 drawSeries()

```
void QwtPlotVectorField::drawSeries (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [override], [virtual]
```

Draw a subset of the points

Parameters

painter	Painter
xMap	Maps x-values into pixel coordinates.
yMap	Maps y-values into pixel coordinates.
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted. If to < 0 the series will be painted to its last sample.

Implements QwtPlotSeriesItem.

Definition at line 807 of file qwt_plot_vectorfield.cpp.

14.104.5.6 drawSymbol()

```
void QwtPlotVectorField::drawSymbol (
    QPainter * painter,
    double x,
    double y,
    double vx,
    double vy ) const  [protected], [virtual]
```

Draw a arrow/symbols at a specific position

x, y, are paint device coordinates, while vx, vy are from the corresponding sample.

See also

setSymbol(), drawSeries()

Definition at line 975 of file qwt_plot_vectorfield.cpp.

14.104.5.7 drawSymbols()

```
void QwtPlotVectorField::drawSymbols (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect,
    int from,
    int to ) const  [protected], [virtual]
```

Draw symbols

Parameters

painter	Painter
xMap	x map
yMap	y map
canvasRect	Contents rectangle of the canvas
from	Index of the first sample to be painted
to	Index of the last sample to be painted

See also

setSymbol(), drawSymbol(), drawSeries()

Definition at line 847 of file qwt_plot_vectorfield.cpp.

14.104.5.8 indicatorOrigin()

```
QwtPlotVectorField::IndicatorOrigin QwtPlotVectorField::indicator↩
Origin ( ) const
```

Returns

origin for the symbols/arrows

Definition at line 401 of file qwt_plot_vectorfield.cpp.

14.104.5.9 legendIcon()

```
QwtGraphic QwtPlotVectorField::legendIcon (
    int index,
    const QSizeF & size ) const  [override], [virtual]
```

Returns

Icon representing the vector fields on the legend

Parameters

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

See also

QwtPlotItem::setLegendIconSize(), QwtPlotItem::legendData()

Reimplemented from QwtPlotItem.

Definition at line 770 of file qwt_plot_vectorfield.cpp.

14.104.5.10 magnitudeRange()

```
QwtInterval QwtPlotVectorField::magnitudeRange ( ) const
```

Returns

min/max magnitudes to be used for color map lookups

See also

setMagnitudeRange(), colorMap()

Definition at line 645 of file qwt_plot_vectorfield.cpp.

14.104.5.11 magnitudeScaleFactor() `double QwtPlotVectorField::magnitudeScaleFactor ( ) const`

Returns

Scale factor used to calculate the arrow length from the magnitude

The length of the arrow in screen coordinate units is calculated by scaling the magnitude by the magnitudeScale↩ Factor.

Default implementation simply scales the vector using the magnitudeScaleFactor property. Re-implement this function to provide special handling for zero/non-zero magnitude arrows, or impose minimum/maximum arrow length limits.

Returns

Length of arrow to be drawn in dependence of vector magnitude.

See also

magnitudeScaleFactor

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 440 of file qwt_plot_vectorfield.cpp.

14.104.5.12 maxArrowLength() `double QwtPlotVectorField::maxArrowLength ( ) const`

Returns

maximum for the arrow length

See also

setMinArrowLength(), maxArrowLength(), arrowLength()

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 705 of file qwt_plot_vectorfield.cpp.

14.104.5.13 minArrowLength() `double QwtPlotVectorField::minArrowLength ( ) const`

Returns

minimum for the arrow length of non zero vectors

See also

setMinArrowLength(), maxArrowLength(), arrowLength()

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 675 of file qwt_plot_vectorfield.cpp.

14.104.5.14 pen() `QPen QwtPlotVectorField::pen ( ) const`

Returns

Pen used to draw the lines

See also

setPen(), brush()

Definition at line 351 of file qwt_plot_vectorfield.cpp.

14.104.5.15 rasterSize() `QSizeF QwtPlotVectorField::rasterSize ( ) const`

Returns

raster size used for filtering samples

See also

setRasterSize(), QwtPlotVectorField::FilterVectors

Definition at line 463 of file qwt_plot_vectorfield.cpp.

14.104.5.16 rtti() `int QwtPlotVectorField::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotField

Reimplemented from QwtPlotItem.

Definition at line 503 of file qwt_plot_vectorfield.cpp.

14.104.5.17 setBrush() `void QwtPlotVectorField::setBrush ( const QBrush & brush )`

Assign a brush.

Parameters

brush	New brush

See also

brush(), pen()

Note

the brush is ignored in MagnitudeAsColor mode

Definition at line 364 of file qwt_plot_vectorfield.cpp.

14.104.5.18 setColorMap() `void QwtPlotVectorField::setColorMap ( QwtColorMap * colorMap )`

Change the color map

The color map is used to map the magnitude of a sample into a color using a known range for the magnitudes.

Parameters

colorMap	Color Map

See also

colorMap(), magnitudeRange()

Definition at line 571 of file qwt_plot_vectorfield.cpp.

14.104.5.19 setIndicatorOrigin() `void QwtPlotVectorField::setIndicatorOrigin ( IndicatorOrigin origin )`

Set the origin for the symbols/arrows

Parameters

origin	Origin

See also

indicatorOrigin()

Definition at line 390 of file qwt_plot_vectorfield.cpp.

14.104.5.20 setMagnitudeMode() void QwtPlotVectorField::setMagnitudeMode (MagnitudeMode *mode*, bool *on* = *true*)

Specify a mode how to represent the magnitude a n arrow/symbol

Parameters

mode	Mode
on	On/Off

See also

testMagnitudeMode()

Definition at line 602 of file qwt_plot_vectorfield.cpp.

14.104.5.21 setMagnitudeRange() void QwtPlotVectorField::setMagnitudeRange (const QwtInterval & *magnitudeRange*)

Sets the min/max magnitudes to be used for color map lookups.

If invalid (min=max=0 or negative values), the range is determined from the current range of magnitudes in the vector samples.

See also

magnitudeRange(), colorMap()

Definition at line 632 of file qwt_plot_vectorfield.cpp.

14.104.5.22 setMagnitudeScaleFactor() void QwtPlotVectorField::setMagnitudeScaleFactor (double *factor*)

Set the magnitudeScaleFactor.

The length of the arrow in screen coordinate units is calculated by scaling the magnitude by the magnitudeScale↩ Factor.

Parameters

factor	Scale factor

See also

magnitudeScaleFactor(), arrowLength()

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 417 of file qwt_plot_vectorfield.cpp.

14.104.5.23 setMaxArrowLength() `void QwtPlotVectorField::setMaxArrowLength ( double length )`

Set a maximum for the arrow length

Parameters

length	Maximum for the arrow length in pixels

See also

maxArrowLength(), setMinArrowLength(), arrowLength()

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 688 of file qwt_plot_vectorfield.cpp.

14.104.5.24 setMinArrowLength() `void QwtPlotVectorField::setMinArrowLength ( double length )`

Set a minimum for the arrow length of non zero vectors

Parameters

length	Minimum for the arrow length in pixels

See also

minArrowLength(), setMaxArrowLength(), arrowLength()

Note

Has no effect when QwtPlotVectorField::MagnitudeAsLength is not enabled

Definition at line 658 of file qwt_plot_vectorfield.cpp.

14.104.5.25 setPaintAttribute() `void QwtPlotVectorField::setPaintAttribute ( PaintAttribute attribute, bool on = true )`

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 475 of file qwt_plot_vectorfield.cpp.

14.104.5.26 setPen() `void QwtPlotVectorField::setPen ( const QPen & pen )`

Assign a pen

Parameters

pen	New pen

See also

pen(), brush()

Note

the pen is ignored in MagnitudeAsColor mode

Definition at line 336 of file qwt_plot_vectorfield.cpp.

14.104.5.27 setRasterSize() `void QwtPlotVectorField::setRasterSize ( const QSizeF & size )`

Set the raster size used for filtering samples

See also

rasterSize(), QwtPlotVectorField::FilterVectors

Definition at line 450 of file qwt_plot_vectorfield.cpp.

14.104.5.28 setSamples() [1/2] `void QwtPlotVectorField::setSamples ( const QVector< QwtVectorFieldSample > & samples )`

Initialize data with an array of samples.

Parameters

samples	Vector of points

Definition at line 541 of file qwt_plot_vectorfield.cpp.

14.104.5.29 setSamples() [2/2] `void QwtPlotVectorField::setSamples ( QwtVectorFieldData * data )`

Assign a series of samples

setSamples() is just a wrapper for setData() without any additional value - beside that it is easier to find for the developer.

Parameters

data	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 556 of file qwt_plot_vectorfield.cpp.

14.104.5.30 setSymbol() `void QwtPlotVectorField::setSymbol ( QwtVectorFieldSymbol * symbol )`

Sets a new arrow symbol (implementation of arrow drawing code).

Parameters

symbol	Arrow symbol

See also

symbol(), drawSymbol()

Note

Ownership is transferred to QwtPlotVectorField.

Definition at line 516 of file qwt_plot_vectorfield.cpp.

14.104.5.31 symbol() `const QwtVectorFieldSymbol * QwtPlotVectorField::symbol ( ) const`

Returns

arrow symbol

See also

setSymbol(), drawSymbol()

Definition at line 532 of file qwt_plot_vectorfield.cpp.

14.104.5.32 testMagnitudeMode() `bool QwtPlotVectorField::testMagnitudeMode ( MagnitudeMode mode ) const`

Returns

True, when mode is enabled

See also

MagnitudeMode, setMagnitudeMode()

Definition at line 619 of file qwt_plot_vectorfield.cpp.

14.104.5.33 testPaintAttribute() `bool QwtPlotVectorField::testPaintAttribute ( PaintAttribute attribute ) const`

Returns

True, when attribute is enabled

See also

PaintAttribute, setPaintAttribute()

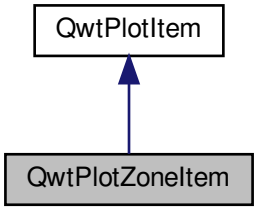
Definition at line 496 of file qwt_plot_vectorfield.cpp.

14.105 QwtPlotZoneItem Class Reference

A plot item, which displays a zone.

`#include <qwt_plot_zoneitem.h>`

Inheritance diagram for QwtPlotZoneItem:



Public Member Functions

- QwtPlotZoneItem ()
  Constructor.
- virtual ~QwtPlotZoneItem ()
  Destructor.
- virtual int rtti () const override
- void setOrientation (Qt::Orientation)
  Set the orientation of the zone.
- Qt::Orientation orientation () const
- void setInterval (double min, double max)
- void setInterval (const QwtInterval &)
- QwtInterval interval () const
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)
  Assign a pen.
- const QPen & pen () const
- void setBrush (const QBrush &)
  Assign a brush.
- const QBrush & brush () const
- virtual void draw (QPainter *, const QwtScaleMap &, const QwtScaleMap &, const QRectF &canvasRect) const override
- virtual QRectF boundingRect () const override

Additional Inherited Members

14.105.1 Detailed Description

A plot item, which displays a zone.

A horizontal zone highlights an interval of the y axis - a vertical zone an interval of the x axis - and is unbounded in the opposite direction. It is filled with a brush and its border lines are optionally displayed with a pen.

Note

For displaying an area that is bounded for x and y coordinates use QwtPlotShapeItem

Definition at line 33 of file qwt_plot_zoneitem.h.

14.105.2 Constructor & Destructor Documentation

14.105.2.1 QwtPlotZoneItem() `QwtPlotZoneItem::QwtPlotZoneItem ( ) [explicit]`

Constructor.

Initializes the zone with no pen and a semi transparent gray brush

Sets the following item attributes:

- QwtPlotItem::AutoScale: false
- QwtPlotItem::Legend: false

The z value is initialized by 5

See also

QwtPlotItem::setItemAttribute(), QwtPlotItem::setZ()

Definition at line 50 of file qwt_plot_zoneitem.cpp.

14.105.3 Member Function Documentation

14.105.3.1 boundingRect() `QRectF QwtPlotZoneItem::boundingRect ( ) const [override], [virtual]`

The bounding rectangle is build from the interval in one direction and something invalid for the opposite direction.

Returns

An invalid rectangle with valid boundaries in one direction

Reimplemented from QwtPlotItem.

Definition at line 297 of file qwt_plot_zoneitem.cpp.

14.105.3.2 brush() `const QBrush & QwtPlotZoneItem::brush ( ) const`

Returns

Brush used to fill the zone

See also

setPen(), brush()

Definition at line 138 of file qwt_plot_zoneitem.cpp.

14.105.3.3 draw()

```
void QwtPlotZoneItem::draw (
    QPainter * painter,
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QRectF & canvasRect ) const [override], [virtual]
```

Draw the zone

Parameters

painter	Painter
xMap	x Scale Map
yMap	y Scale Map
canvasRect	Contents rectangle of the canvas in painter coordinates

Implements QwtPlotItem.

Definition at line 223 of file qwt_plot_zoneitem.cpp.

14.105.3.4 interval() `QwtInterval QwtPlotZoneItem::interval ( ) const`

Returns

Zone interval

See also

setInterval(), orientation()

Definition at line 209 of file qwt_plot_zoneitem.cpp.

14.105.3.5 orientation() `Qt::Orientation QwtPlotZoneItem::orientation ( ) const`

Returns

Orientation of the zone

See also

setOrientation()

Definition at line 165 of file qwt_plot_zoneitem.cpp.

14.105.3.6 pen() `const QPen & QwtPlotZoneItem::pen ( ) const`

Returns

Pen used to draw the border lines

See also

setPen(), brush()

Definition at line 112 of file qwt_plot_zoneitem.cpp.

14.105.3.7 rtti() `int QwtPlotZoneItem::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PlotZone

Reimplemented from QwtPlotItem.

Definition at line 68 of file qwt_plot_zoneitem.cpp.

14.105.3.8 setBrush() `void QwtPlotZoneItem::setBrush ( const QBrush & brush )`

Assign a brush.

The brush is used to fill the zone

Parameters

brush	Brush

See also

pen(), setBrush()

Definition at line 125 of file qwt_plot_zoneitem.cpp.

14.105.3.9 setInterval() [1/2]

```
void QwtPlotZoneItem::setInterval (
    const QwtInterval & interval )
```

Set the interval of the zone

For a horizontal zone the interval is related to the y axis, for a vertical zone it is related to the x axis.

Parameters

interval	Zone interval

See also

interval(), setOrientation()

Definition at line 196 of file qwt_plot_zoneitem.cpp.

14.105.3.10 setInterval() [2/2]

```
void QwtPlotZoneItem::setInterval (
    double min,
    double max )
```

Set the interval of the zone

For a horizontal zone the interval is related to the y axis, for a vertical zone it is related to the x axis.

Parameters

min	Minimum of the interval
max	Maximum of the interval

See also

interval(), setOrientation()

Definition at line 181 of file qwt_plot_zoneitem.cpp.

14.105.3.11 setOrientation() `void QwtPlotZoneItem::setOrientation ( Qt::Orientation orientation )`

Set the orientation of the zone.

A horizontal zone highlights an interval of the y axis, a vertical zone of the x axis. It is unbounded in the opposite direction.

See also

orientation(), QwtPlotItem::setAxes()

Definition at line 152 of file qwt_plot_zoneitem.cpp.

14.105.3.12 setPen() [1/2] `void QwtPlotZoneItem::setPen ( const QColor & color, qreal width = 0.0, Qt::PenStyle style = Qt::SolidLine )`

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 86 of file qwt_plot_zoneitem.cpp.

14.105.3.13 setPen() [2/2] `void QwtPlotZoneItem::setPen ( const QPen & pen )`

Assign a pen.

The pen is used to draw the border lines of the zone

Parameters

pen	Pen

See also

pen(), setBrush()

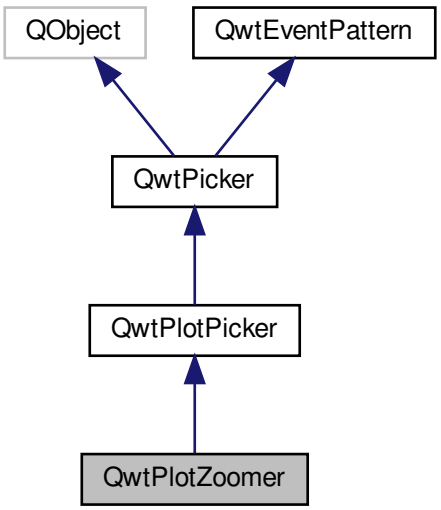
Definition at line 99 of file qwt_plot_zoneitem.cpp.

14.106 QwtPlotZoomer Class Reference

QwtPlotZoomer provides stacked zooming for a plot widget.

```
#include <qwt_plot_zoomer.h>
```

Inheritance diagram for QwtPlotZoomer:



Public Slots

- void moveBy (double dx, double dy)
- virtual void moveTo (const QPointF &)
- virtual void zoom (const QRectF &)

  Zoom in.
- virtual void zoom (int offset)

  Zoom in or out.

Signals

- void zoomed (const QRectF &rect)

Public Member Functions

- QwtPlotZoomer (QWidget *, bool doReplot=true)

  Create a zoomer for a plot canvas.
- QwtPlotZoomer (QwtAxisId xAxis, QwtAxisId yAxis, QWidget *, bool doReplot=true)

  Create a zoomer for a plot canvas.
- virtual void setZoomBase (bool doReplot=true)
- virtual void setZoomBase (const QRectF &)

  Set the initial size of the zoomer.
- QRectF zoomBase () const
- QRectF zoomRect () const
- virtual void setAxes (QwtAxisId xAxis, QwtAxisId yAxis) override
- void setMaxStackDepth (int)

  Limit the number of recursive zoom operations to depth.
- int maxStackDepth () const
- const QStack< QRectF > & zoomStack () const
- void setZoomStack (const QStack< QRectF > &, int zoomRectIndex=-1)

  Assign a zoom stack.
- uint zoomRectIndex () const

Protected Member Functions

- virtual void rescale ()
- virtual QSizeF minZoomSize () const

  Limit zooming by a minimum rectangle.
- virtual void widgetMouseReleaseEvent (QMouseEvent *) override
- virtual void widgetKeyPressEvent (QKeyEvent *) override
- virtual void begin () override
- virtual bool end (bool ok=true) override
- virtual bool accept (QPolygon &) const override

  Check and correct a selected rectangle.

Additional Inherited Members

14.106.1 Detailed Description

QwtPlotZoomer provides stacked zooming for a plot widget.

QwtPlotZoomer selects rectangles from user inputs (mouse or keyboard) translates them into plot coordinates and adjusts the axes to them. The selection is supported by a rubber band and optionally by displaying the coordinates of the current mouse position.

Zooming can be repeated as often as possible, limited only by maxStackDepth() or minZoomSize(). Each rectangle is pushed on a stack.

The default setting how to select rectangles is a QwtPickerDragRectMachine with the following bindings:

- QwtEventPattern::MouseSelect1
  The first point of the zoom rectangle is selected by a mouse press, the second point from the position, where the mouse is released.

- QwtEventPattern::KeySelect1
  The first key press selects the first, the second key press selects the second point.
- QwtEventPattern::KeyAbort
  Discard the selection in the state, where the first point is selected.

To traverse the zoom stack the following bindings are used:

- QwtEventPattern::MouseSelect3, QwtEventPattern::KeyUndo
  Zoom out one position on the zoom stack
- QwtEventPattern::MouseSelect6, QwtEventPattern::KeyRedo
  Zoom in one position on the zoom stack
- QwtEventPattern::MouseSelect2, QwtEventPattern::KeyHome
  Zoom to the zoom base

The setKeyPattern() and setMousePattern() functions can be used to configure the zoomer actions. The following example shows, how to configure the 'I' and 'O' keys for zooming in and out one position on the zoom stack. The "Home" key is used to "unzoom" the plot.

```
zoomer = new QwtPlotZoomer( plot );
zoomer->setKeyPattern( QwtEventPattern::KeyRedo, Qt::Key_I, Qt::ShiftModifier );
zoomer->setKeyPattern( QwtEventPattern::KeyUndo, Qt::Key_O, Qt::ShiftModifier );
zoomer->setKeyPattern( QwtEventPattern::KeyHome, Qt::Key_Home );
```

QwtPlotZoomer is tailored for plots with one x and y axis, but it is allowed to attach a second QwtPlotZoomer (without rubber band and tracker) for the other axes.

Note

The realtime example includes an derived zoomer class that adds scrollbars to the plot canvas.

See also

QwtPlotPanner, QwtPlotMagnifier

Definition at line 79 of file qwt_plot_zoomer.h.

14.106.2 Constructor & Destructor Documentation

14.106.2.1 QwtPlotZoomer() [1/2]

```
QwtPlotZoomer::QwtPlotZoomer (
    QWidget * 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 QwtAxis::XBottom. If both or no y-axis are enabled, it is set to QwtAxis::YLeft.

The zoomer is initialized with a QwtPickerDragRectMachine, the tracker mode is set to QwtPicker::ActiveOnly and the rubber band is set to QwtPicker::RectRubberBand

Parameters

canvas	Plot canvas to observe, also the parent object
doReplot	Call QwtPlot::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()

Definition at line 109 of file qwt_plot_zoomer.cpp.

14.106.2.2 QwtPlotZoomer() [2/2]

```
QwtPlotZoomer::QwtPlotZoomer (
    QwtAxisId xAxisId,
    QwtAxisId yAxisId,
    QWidget * canvas,
    bool doReplot = true )  [explicit]
```

Create a zoomer for a plot canvas.

The zoomer is initialized with a QwtPickerDragRectMachine, the tracker mode is set to QwtPicker::ActiveOnly and the rubber band is set to QwtPicker::RectRubberBand

Parameters

xAxisId	X axis of the zoomer
yAxisId	Y axis of the zoomer
canvas	Plot canvas to observe, also the parent object
doReplot	Call QwtPlot::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()

Definition at line 133 of file qwt_plot_zoomer.cpp.

14.106.3 Member Function Documentation

14.106.3.1 accept()

```
bool QwtPlotZoomer::accept (
    QPolygon & pa ) const  [override], [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 the rectangle is accepted, or has been changed to an accepted one.

Reimplemented from QwtPicker.

Definition at line 567 of file qwt_plot_zoomer.cpp.

14.106.3.2 begin() `void QwtPlotZoomer::begin ( ) [override], [protected], [virtual]`

Rejects selections, when the stack depth is too deep, or the zoomed rectangle is minZoomSize().

See also

minZoomSize(), maxStackDepth()

Reimplemented from QwtPicker.

Definition at line 609 of file qwt_plot_zoomer.cpp.

14.106.3.3 end() `bool QwtPlotZoomer::end ( bool ok = true ) [override], [protected], [virtual]`

Expand the selected rectangle to minZoomSize() and zoom in if accepted.

Parameters

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

See also

accept(), minZoomSize()

Returns

True if the selection has been accepted, false otherwise

Reimplemented from QwtPlotPicker.

Definition at line 643 of file qwt_plot_zoomer.cpp.

14.106.3.4 maxStackDepth() `int QwtPlotZoomer::maxStackDepth ( ) const`

Returns

Maximal depth of the zoom stack.

See also

setMaxStackDepth()

Definition at line 201 of file qwt_plot_zoomer.cpp.

14.106.3.5 minZoomSize() `QSizeF QwtPlotZoomer::minZoomSize ( ) const [protected], [virtual]`

Limit zooming by a minimum rectangle.

Returns

zoomBase().width() / 10e4, zoomBase().height() / 10e4

Definition at line 597 of file qwt_plot_zoomer.cpp.

14.106.3.6 moveBy `void QwtPlotZoomer::moveBy ( double dx, double dy ) [slot]`

Move the current zoom rectangle.

Parameters

dx	X offset
dy	Y offset

Note

The changed rectangle is limited by the zoom base

Definition at line 520 of file qwt_plot_zoomer.cpp.

14.106.3.7 moveTo `void QwtPlotZoomer::moveTo ( const QPointF & pos ) [virtual], [slot]`

Move the the current zoom rectangle.

Parameters

pos	New position

See also

QRectF::moveTo()

Note

The changed rectangle is limited by the zoom base

Definition at line 534 of file qwt_plot_zoomer.cpp.

14.106.3.8 rescale() `void QwtPlotZoomer::rescale ( ) [protected], [virtual]`

Adjust the observed plot to zoomRect()

Note

Initiates QwtPlot::replot()

Definition at line 416 of file qwt_plot_zoomer.cpp.

14.106.3.9 setAxes() `void QwtPlotZoomer::setAxes ( QwtAxisId xAxisId, QwtAxisId yAxisId ) [override], [virtual]`

Reinitialize the axes, and set the zoom base to their scales.

Parameters

x↩ AxisId	X axis
y↩ AxisId	Y axis

Reimplemented from QwtPlotPicker.

Definition at line 455 of file qwt_plot_zoomer.cpp.

14.106.3.10 setMaxStackDepth() `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

depth	Maximum for the stack depth

See also

maxStackDepth()

Note

depth doesn't include the zoom base, so zoomStack().count() might be maxStackDepth() + 1.

Definition at line 174 of file qwt_plot_zoomer.cpp.

14.106.3.11 setZoomBase() [1/2] void QwtPlotZoomer::setZoomBase (bool *doReplot* = *true*) [virtual]

Reinitialized the zoom stack with scaleRect() as base.

Parameters

doReplot	Call QwtPlot::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().

Definition at line 235 of file qwt_plot_zoomer.cpp.

14.106.3.12 setZoomBase() [2/2] void QwtPlotZoomer::setZoomBase (const QRectF & *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

base	Zoom base

See also

zoomBase(), scaleRect()

Definition at line 261 of file qwt_plot_zoomer.cpp.

14.106.3.13 setZoomStack() `void QwtPlotZoomer::setZoomStack ( const QStack< QRectF > & zoomStack, 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

zoomStack	New zoom stack
zoomRectIndex	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()

Definition at line 383 of file qwt_plot_zoomer.cpp.

14.106.3.14 widgetKeyPressEvent() `void QwtPlotZoomer::widgetKeyPressEvent ( QKeyEvent * ke ) [override], [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.

Definition at line 497 of file qwt_plot_zoomer.cpp.

14.106.3.15 widgetMouseReleaseEvent() `void QwtPlotZoomer::widgetMouseReleaseEvent ( QMouseEvent * me ) [override], [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.

Definition at line 474 of file qwt_plot_zoomer.cpp.

14.106.3.16 zoom [1/2] `void QwtPlotZoomer::zoom ( const QRectF & rect ) [virtual], [slot]`

Zoom in.

Clears all rectangles above the current position of the zoom stack and pushes the normalized rectangle on it.

Note

If the maximal stack depth is reached, zoom is ignored.

The zoomed signal is emitted.

Definition at line 310 of file qwt_plot_zoomer.cpp.

14.106.3.17 zoom [2/2] `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

offset	Offset relative to the current position of the zoom stack.

Note

The zoomed signal is emitted.

See also

zoomRectIndex()

Definition at line 347 of file qwt_plot_zoomer.cpp.

14.106.3.18 zoomBase() `QRectF QwtPlotZoomer::zoomBase ( ) const`

Returns

Initial rectangle of the zoomer

See also

setZoomBase(), zoomRect()

Definition at line 221 of file qwt_plot_zoomer.cpp.

14.106.3.19 zoomed `void QwtPlotZoomer::zoomed ( const QRectF & rect ) [signal]`

A signal emitting the zoomRect(), when the plot has been zoomed in or out.

Parameters

rect	Current zoom rectangle.

14.106.3.20 zoomRect() `QRectF QwtPlotZoomer::zoomRect ( ) const`

Returns

Rectangle at the current position on the zoom stack.

See also

zoomRectIndex(), scaleRect().

Definition at line 287 of file qwt_plot_zoomer.cpp.

14.106.3.21 zoomRectIndex() `uint QwtPlotZoomer::zoomRectIndex ( ) const`

Returns

Index of current position of zoom stack.

Definition at line 295 of file qwt_plot_zoomer.cpp.

14.106.3.22 zoomStack() `const QStack< QRectF > & QwtPlotZoomer::zoomStack ( ) const`

Returns

The zoom stack. zoomStack()[0] is the zoom base, zoomStack()[1] the first zoomed rectangle.

See also

setZoomStack(), zoomRectIndex()

Definition at line 212 of file qwt_plot_zoomer.cpp.

14.107 QwtPoint3D Class Reference

QwtPoint3D class defines a 3D point in double coordinates.

`#include <qwt_point_3d.h>`

Public Member Functions

- QwtPoint3D ()
- QwtPoint3D (double x, double y, double z)

  Constructs a point with coordinates specified by x, y and z.
- QwtPoint3D (const QPointF &)
- bool isNull () const
- double x () const
- double y () const
- double z () const
- double & rx ()
- double & ry ()
- double & rz ()
- void setX (double x)

  Sets the x-coordinate of the point to the value specified by x.
- void setY (double y)

  Sets the y-coordinate of the point to the value specified by y.
- void setZ (double y)

  Sets the z-coordinate of the point to the value specified by z.
- QPointF toPoint () const
- bool operator== (const QwtPoint3D &) const
- bool operator!= (const QwtPoint3D &) const

14.107.1 Detailed Description

QwtPoint3D class defines a 3D point in double coordinates.

Definition at line 22 of file qwt_point_3d.h.

14.107.2 Constructor & Destructor Documentation

14.107.2.1 QwtPoint3D() [1/2] QwtPoint3D::QwtPoint3D ( ) [inline]

Constructs a null point.

See also

isNull()

Definition at line 65 of file qwt_point_3d.h.

14.107.2.2 QwtPoint3D() [2/2] QwtPoint3D::QwtPoint3D (const QPointF & *other*) [inline]

Constructs a point with x and y coordinates from a 2D point, and a z coordinate of 0.

Definition at line 84 of file qwt_point_3d.h.

14.107.3 Member Function Documentation

14.107.3.1 isNull() bool QwtPoint3D::isNull ( ) const [inline]

Returns

True if the point is null; otherwise returns false.

A point is considered to be null if x, y and z-coordinates are equal to zero.

Definition at line 97 of file qwt_point_3d.h.

14.107.3.2 operator"!=() bool QwtPoint3D::operator!= (const QwtPoint3D & *other*) const [inline]

Returns

True if this rect and other are different; otherwise returns false.

Definition at line 171 of file qwt_point_3d.h.

14.107.3.3 operator==() bool QwtPoint3D::operator== (const QwtPoint3D & *other*) const [inline]

Returns

True, if this point and other are equal; otherwise returns false.

Definition at line 165 of file qwt_point_3d.h.

14.107.3.4 rx() double & QwtPoint3D::rx ( ) [inline]

Returns

A reference to the x-coordinate of the point.

Definition at line 121 of file qwt_point_3d.h.

14.107.3.5 ry() double & QwtPoint3D::ry ( ) [inline]

Returns

A reference to the y-coordinate of the point.

Definition at line 127 of file qwt_point_3d.h.

14.107.3.6 rz() double & QwtPoint3D::rz ( ) [inline]

Returns

A reference to the z-coordinate of the point.

Definition at line 133 of file qwt_point_3d.h.

14.107.3.7 toPoint() `QPointF QwtPoint3D::toPoint ( ) const [inline]`

Returns

2D point, where the z coordinate is dropped.

Definition at line 159 of file qwt_point_3d.h.

14.107.3.8 x() `double QwtPoint3D::x ( ) const [inline]`

Returns

The x-coordinate of the point.

Definition at line 103 of file qwt_point_3d.h.

14.107.3.9 y() `double QwtPoint3D::y ( ) const [inline]`

Returns

The y-coordinate of the point.

Definition at line 109 of file qwt_point_3d.h.

14.107.3.10 z() `double QwtPoint3D::z ( ) const [inline]`

Returns

The z-coordinate of the point.

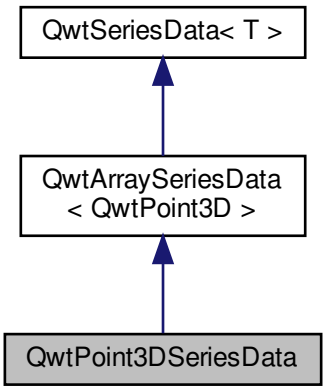
Definition at line 115 of file qwt_point_3d.h.

14.108 QwtPoint3DSeriesData Class Reference

Interface for iterating over an array of 3D points.

`#include <qwt_series_data.h>`

Inheritance diagram for QwtPoint3DSeriesData:



Public Member Functions

- QwtPoint3DSeriesData (const QVector< QwtPoint3D > &=QVector< QwtPoint3D >())
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.

Additional Inherited Members

14.108.1 Detailed Description

Interface for iterating over an array of 3D points.

Definition at line 219 of file qwt_series_data.h.

14.108.2 Constructor & Destructor Documentation

14.108.2.1 QwtPoint3DSeriesData() `QwtPoint3DSeriesData::QwtPoint3DSeriesData ( const QVector< QwtPoint3D > & samples = QVector< QwtPoint3D >( ) )`

Constructor

Parameters

samples	Samples

Definition at line 275 of file qwt_series_data.cpp.

14.108.3 Member Function Documentation

14.108.3.1 boundingRect() `QRectF QwtPoint3DSeriesData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

Implements QwtSeriesData< T >.

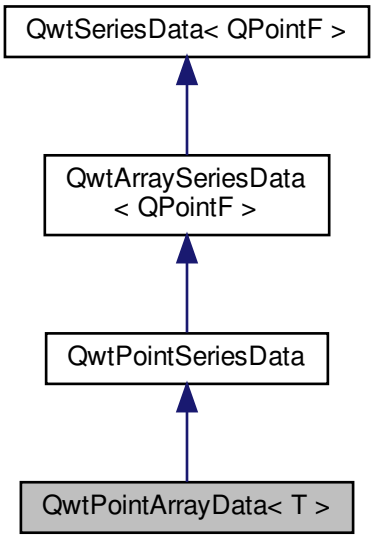
Definition at line 289 of file qwt_series_data.cpp.

14.109 QwtPointArrayData< T > Class Template Reference

Interface for iterating over two QVector<T> objects.

`#include <qwt_point_data.h>`

Inheritance diagram for QwtPointArrayData< T >:



Public Member Functions

- QwtPointArrayData (const QVector< T > &x, const QVector< T > &y)
- QwtPointArrayData (const T *x, const T *y, size_t size)
- virtual size_t size () const override
- virtual QPointF sample (size_t index) const override
- const QVector< T > & xData () const
- const QVector< T > & yData () const

Additional Inherited Members

14.109.1 Detailed Description

template<typename T>
class QwtPointArrayData< T >

Interface for iterating over two QVector<T> objects.

Definition at line 22 of file qwt_point_data.h.

14.109.2 Constructor & Destructor Documentation

14.109.2.1 QwtPointArrayData() [1/2] template<typename T >

```
QwtPointArrayData< T >::QwtPointArrayData (
            const QVector< T > & x,
            const QVector< T > & y )
```

Constructor

Parameters

x	Array of x values
y	Array of y values

See also

QwtPlotCurve::setData(), QwtPlotCurve::setSamples()

Definition at line 200 of file qwt_point_data.h.

14.109.2.2 QwtPointArrayData() [2/2] template<typename T >

```
QwtPointArrayData< T >::QwtPointArrayData (
            const T * x,
            const T * y,
            size_t size )
```

Constructor

Parameters

x	Array of x values
y	Array of y values
size	Size of the x and y arrays

See also

QwtPlotCurve::setData(), QwtPlotCurve::setSamples()

Definition at line 216 of file qwt_point_data.h.

14.109.3 Member Function Documentation

14.109.3.1 sample() `template<typename T >`

```
QPointF QwtPointArrayData< T >::sample (
            size_t index ) const  [override], [virtual]
```

Return the sample at position i

Parameters

index	Index

Returns

Sample at position i

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 239 of file qwt_point_data.h.

14.109.3.2 size() `template<typename T >`

```
size_t QwtPointArrayData< T >::size  [override], [virtual]
```

Returns

Size of the data set

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 227 of file qwt_point_data.h.

14.109.3.3 xData() template<typename T >

const QVector< T > & QwtPointArrayData< T >::xData

Returns

Array of the x-values

Definition at line 246 of file qwt_point_data.h.

14.109.3.4 yData() template<typename T >

const QVector< T > & QwtPointArrayData< T >::yData

Returns

Array of the y-values

Definition at line 253 of file qwt_point_data.h.

14.110 QwtPointMapper Class Reference

A helper class for translating a series of points.

#include <qwt_point_mapper.h>

Public Types

- enum TransformationFlag { RoundPoints = 0x01 , WeedOutPoints = 0x02 , WeedOutIntermediatePoints = 0x04 }

  Flags affecting the transformation process.
- typedef QFlags< TransformationFlag > TransformationFlags

Public Member Functions

- QwtPointMapper ()

  Constructor.
- ~QwtPointMapper ()

  Destructor.
- void setFlags (TransformationFlags)
- TransformationFlags flags () const
- void setFlag (TransformationFlag, bool on=true)
- bool testFlag (TransformationFlag) const
- void setBoundingRect (const QRectF &)
- QRectF boundingRect () const
- QPolygonF toPolygonF (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtSeriesData< QPointF > *series, int from, int to) const

  Translate a series of points into a QPolygonF.
- QPolygon toPolygon (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtSeriesData< QPointF > *series, int from, int to) const

Translate a series of points into a QPolygon.

- QPolygon toPoints (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtSeriesData< QPointF > *series, int from, int to) const

  Translate a series of points into a QPolygon.
- QPolygonF toPointsF (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtSeriesData< QPointF > *series, int from, int to) const

  Translate a series into a QPolygonF.
- QImage toImage (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtSeriesData< QPointF > *series, int from, int to, const QPen &, bool antialiased, uint numThreads) const

  Translate a series into a QImage.

14.110.1 Detailed Description

A helper class for translating a series of points.

QwtPointMapper is a collection of methods and optimizations for translating a series of points into paint device coordinates. It is used by QwtPlotCurve but might also be useful for similar plot items displaying a QwtSeriesData<QPointF>.

Definition at line 32 of file qwt_point_mapper.h.

14.110.2 Member Typedef Documentation

14.110.2.1 TransformationFlags `typedef QFlags<TransformationFlag > QwtPointMapper::TransformationFlags`

An ORed combination of TransformationFlag values.

Definition at line 71 of file qwt_point_mapper.h.

14.110.3 Member Enumeration Documentation

14.110.3.1 TransformationFlag `enum QwtPointMapper::TransformationFlag`

Flags affecting the transformation process.

Enumerator

See also

setFlag(), setFlags()

Enumerator

RoundPoints	Round points to integer values.
WeedOutPoints	Try to remove points, that are translated to the same position.
WeedOutIntermediatePoints	An even more aggressive weeding algorithm, that can be used in toPolygon(). A consecutive chunk of points being mapped to the same x coordinate is reduced to 4 points: • first point • point with the minimum y coordinate • point with the maximum y coordinate • last point In the worst case (first and last points are never one of the extremes) the number of points will be 4 times the width. As the algorithm is fast it can be used inside of a polyline render cycle.

Definition at line 39 of file qwt_point_mapper.h.

14.110.4 Member Function Documentation

14.110.4.1 boundingRect() `QRectF QwtPointMapper::boundingRect ( ) const`

Returns

Bounding rectangle

See also

setBoundingRect()

Definition at line 621 of file qwt_point_mapper.cpp.

14.110.4.2 flags() `QwtPointMapper::TransformationFlags QwtPointMapper::flags ( ) const`

Returns

Flags affecting the transformation process

See also

setFlags(), setFlag()

Definition at line 573 of file qwt_point_mapper.cpp.

14.110.4.3 setBoundingRect() `void QwtPointMapper::setBoundingRect ( const QRectF & rect )`

Set a bounding rectangle for the point mapping algorithm

A valid bounding rectangle can be used for optimizations

Parameters

rect	Bounding rectangle

See also

boundingRect()

Definition at line 612 of file qwt_point_mapper.cpp.

14.110.4.4 setFlag() `void QwtPointMapper::setFlag ( TransformationFlag flag, bool on = true )`

Modify a flag affecting the transformation process

Parameters

flag	Flag type
on	Value

See also

flag(), setFlags()

Definition at line 586 of file qwt_point_mapper.cpp.

14.110.4.5 setFlags() void QwtPointMapper::setFlags (TransformationFlags *flags*)

Set the flags affecting the transformation process

Parameters

flags	Flags

See also

flags(), setFlag()

Definition at line 564 of file qwt_point_mapper.cpp.

14.110.4.6 testFlag() bool QwtPointMapper::testFlag (TransformationFlag *flag*) const

Returns

True, when the flag is set

Parameters

flag	Flag type

See also

setFlag(), setFlags()

Definition at line 599 of file qwt_point_mapper.cpp.

14.110.4.7 toImage() QImage QwtPointMapper::toImage (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QwtSeriesData< QPointF > * *series*, int *from*, int *to*, const QPen & *pen*, bool *antialiased*, uint *numThreads*) const

Translate a series into a QImage.

Parameters

xMap	x map

Parameters

yMap	y map
series	Series of points to be mapped
from	Index of the first point to be painted
to	Index of the last point to be painted
pen	Pen used for drawing a point of the image, where a point is mapped to
antialiased	True, when the dots should be displayed antialiased
numThreads	Number of threads to be used for rendering. If numThreads is set to 0, the system specific ideal thread count is used.

Returns

Image displaying the series

Definition at line 883 of file qwt_point_mapper.cpp.

14.110.4.8 toPoints()

```
QPolygon QwtPointMapper::toPoints (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QwtSeriesData< QPointF > * series,
    int from,
    int to ) const
```

Translate a series of points into a QPolygon.

- WeedOutPoints & boundingRect().isValid() All points that are mapped to the same position will be one point. Points outside of the bounding rectangle are ignored.
- WeedOutPoints & !boundingRect().isValid() All consecutive points that are mapped to the same position will one point
- !WeedOutPoints & boundingRect().isValid() Points outside of the bounding rectangle are ignored.

Parameters

xMap	x map
yMap	y map
series	Series of points to be mapped
from	Index of the first point to be painted
to	Index of the last point to be painted

Returns

Translated polygon

Definition at line 833 of file qwt_point_mapper.cpp.

14.110.4.9 toPointsF()

```
QPolygonF QwtPointMapper::toPointsF (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QwtSeriesData< QPointF > * series,
    int from,
    int to ) const
```

Translate a series into a QPolygonF.

- WeedOutPoints & RoundPoints & boundingRect().isValid() All points that are mapped to the same position will be one point. Points outside of the bounding rectangle are ignored.
- WeedOutPoints & RoundPoints & !boundingRect().isValid() All consecutive points that are mapped to the same position will one point
- WeedOutPoints & !RoundPoints All consecutive points that are mapped to the same position will one point
- !WeedOutPoints & boundingRect().isValid() Points outside of the bounding rectangle are ignored.

When RoundPoints is set all points are rounded to integers but returned as PolygonF - what only makes sense when the further processing of the values need a QPolygonF.

Parameters

xMap	x map
yMap	y map
series	Series of points to be mapped
from	Index of the first point to be painted
to	Index of the last point to be painted

Returns

Translated polygon

Definition at line 759 of file qwt_point_mapper.cpp.

14.110.4.10 toPolygon()

```
QPolygon QwtPointMapper::toPolygon (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QwtSeriesData< QPointF > * series,
    int from,
    int to ) const
```

Translate a series of points into a QPolygon.

When the WeedOutPoints flag is enabled consecutive points, that are mapped to the same position will be one point.

Parameters

xMap	x map
yMap	y map
series	Series of points to be mapped
from	Index of the first point to be painted
to	Index of the last point to be painted

Returns

Translated polygon

Definition at line 702 of file qwt_point_mapper.cpp.

14.110.4.11 toPolygonF()

```
QPolygonF QwtPointMapper::toPolygonF (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QwtSeriesData< QPointF > * series,
    int from,
    int to ) const
```

Translate a series of points into a QPolygonF.

When the WeedOutPoints flag is enabled consecutive points, that are mapped to the same position will be one point.

When RoundPoints is set all points are rounded to integers but returned as PolygonF - what only makes sense when the further processing of the values need a QPolygonF.

When RoundPoints & WeedOutIntermediatePoints is enabled an even more aggressive weeding algorithm is enabled.

Parameters

xMap	x map
yMap	y map
series	Series of points to be mapped
from	Index of the first point to be painted
to	Index of the last point to be painted

Returns

Translated polygon

Definition at line 647 of file qwt_point_mapper.cpp.

14.111 QwtPointPolar Class Reference

A point in polar coordinates.

```
#include <qwt_point_polar.h>
```

Public Member Functions

- QwtPointPolar ()
- QwtPointPolar (double azimuth, double radius)
- QwtPointPolar (const QPointF &)
- void setPoint (const QPointF &)
- QPointF toPoint () const
- bool isValid () const

  Returns true if radius() >= 0.0.
- bool isNull () const

  Returns true if radius() >= 0.0.
- double radius () const

  Returns the radius.
- double azimuth () const

  Returns the azimuth.
- double & rRadius ()

  Returns the radius.
- double & rAzimuth ()

  Returns the azimuth.
- void setRadius (double)

  Sets the radius to radius.
- void setAzimuth (double)

  Sets the azimuth to azimuth.
- bool operator== (const QwtPointPolar &) const

  Compare 2 points.
- bool operator!= (const QwtPointPolar &) const
- QwtPointPolar normalized () const

14.111.1 Detailed Description

A point in polar coordinates.

In polar coordinates a point is determined by an angle and a distance. See http://en.wikipedia.org/wiki/Polar_coordinate_system

Definition at line 28 of file qwt_point_polar.h.

14.111.2 Constructor & Destructor Documentation

14.111.2.1 QwtPointPolar() [1/3] QwtPointPolar::QwtPointPolar ( ) [inline]

Constructs a null point, with a radius and azimuth set to 0.0.

See also

QPointF::isNull()

Definition at line 71 of file qwt_point_polar.h.

14.111.2.2 QwtPointPolar() [2/3] QwtPointPolar::QwtPointPolar (double *azimuth*, double *radius*) [inline]

Constructs a point with coordinates specified by radius and azimuth.

Parameters

azimuth	Azimuth
radius	Radius

Definition at line 83 of file qwt_point_polar.h.

14.111.2.3 QwtPointPolar() [3/3] QwtPointPolar::QwtPointPolar (const QPointF & p)

Convert and assign values from a point in Cartesian coordinates

Parameters

p	Point in Cartesian coordinates

See also

setPoint(), toPoint()

Definition at line 44 of file qwt_point_polar.cpp.

14.111.3 Member Function Documentation

14.111.3.1 normalized() QwtPointPolar QwtPointPolar::normalized ( ) const

Normalize radius and azimuth

When the radius is $<$ 0.0 it is set to 0.0. The azimuth is a value $>=$ 0.0 and $<$ 2 $*$ M_PI.

Returns

Normalized point

Definition at line 118 of file qwt_point_polar.cpp.

14.111.3.2 operator"!=() bool QwtPointPolar::operator!= (const QwtPointPolar & *other*) const

Compare 2 points

Two points are equal to each other if radius and azimuth-coordinates are the same. Points are not equal, when the azimuth differs, but other.azimuth() == azimuth() % (2 * PI).

Returns

True if the point is not equal to other; otherwise return false.

See also

normalized()

Definition at line 105 of file qwt_point_polar.cpp.

14.111.3.3 operator==() bool QwtPointPolar::operator== (const QwtPointPolar & *other*) const

Compare 2 points.

Two points are equal to each other if radius and azimuth-coordinates are the same. Points are not equal, when the azimuth differs, but other.azimuth() == azimuth() % (2 * PI).

Returns

True if the point is equal to other; otherwise return false.

See also

normalized()

Definition at line 90 of file qwt_point_polar.cpp.

14.111.3.4 setPoint() void QwtPointPolar::setPoint (const QPointF & *p*)

Convert and assign values from a point in Cartesian coordinates

Parameters

p	Point in Cartesian coordinates

Definition at line 54 of file qwt_point_polar.cpp.

14.111.3.5 toPoint() `QPointF QwtPointPolar::toPoint ( ) const`

Convert and return values in Cartesian coordinates

Returns

Converted point in Cartesian coordinates

Note

Invalid or null points will be returned as QPointF(0.0, 0.0)

See also

isValid(), isNull()

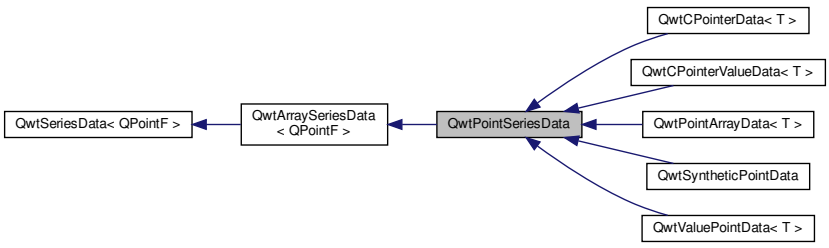
Definition at line 68 of file qwt_point_polar.cpp.

14.112 QwtPointSeriesData Class Reference

Interface for iterating over an array of points.

`#include <qwt_series_data.h>`

Inheritance diagram for QwtPointSeriesData:



Public Member Functions

- QwtPointSeriesData (const QVector< QPointF > &=QVector< QPointF >())
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.

Additional Inherited Members

14.112.1 Detailed Description

Interface for iterating over an array of points.

Definition at line 209 of file qwt_series_data.h.

14.112.2 Constructor & Destructor Documentation

14.112.2.1 QwtPointSeriesData() `QwtPointSeriesData::QwtPointSeriesData ( const QVector< QPointF > & samples = QVector< QPointF >( ) )`

Constructor

Parameters

samples	Samples

Definition at line 250 of file qwt_series_data.cpp.

14.112.3 Member Function Documentation

14.112.3.1 boundingRect() `QRectF QwtPointSeriesData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

Implements QwtSeriesData< QPointF >.

Reimplemented in QwtSyntheticPointData.

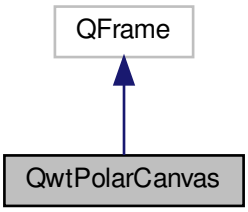
Definition at line 263 of file qwt_series_data.cpp.

14.113 QwtPolarCanvas Class Reference

Canvas of a QwtPolarPlot.

`#include <qwt_polar_canvas.h>`

Inheritance diagram for QwtPolarCanvas:



Public Types

- enum PaintAttribute { BackingStore = 0x01 }
  Paint attributes.
- typedef QFlags< PaintAttribute > PaintAttributes

Public Member Functions

- QwtPolarCanvas (QwtPolarPlot *)
  Constructor.
- virtual ~QwtPolarCanvas ()
  Destructor.
- QwtPolarPlot * plot ()
- const QwtPolarPlot * plot () const
- void setPaintAttribute (PaintAttribute, bool on=true)
  Changing the paint attributes.
- bool testPaintAttribute (PaintAttribute) const
- const QPixmap * backingStore () const
- void invalidateBackingStore ()
  Invalidate the internal backing store.
- QwtPointPolar invTransform (const QPoint &) const
- QPoint transform (const QwtPointPolar &) const

Protected Member Functions

- virtual void paintEvent (QPaintEvent *) override
- virtual void resizeEvent (QResizeEvent *) override

14.113.1 Detailed Description

Canvas of a QwtPolarPlot.

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

Note

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

See also

QwtPolarPlot

Definition at line 27 of file qwt_polar_canvas.h.

14.113.2 Member Typedef Documentation

14.113.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPolarCanvas::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 50 of file qwt_polar_canvas.h.

14.113.3 Member Enumeration Documentation

14.113.3.1 PaintAttribute `enum QwtPolarCanvas::PaintAttribute`

Paint attributes.

The default setting enables BackingStore

See also

setPaintAttribute(), testPaintAttribute(), backingStore()

Enumerator

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

Definition at line 40 of file qwt_polar_canvas.h.

14.113.4 Member Function Documentation

14.113.4.1 backingStore() `const QPixmap * QwtPolarCanvas::backingStore ( ) const`

Returns

Backing store, might be null

Definition at line 170 of file qwt_polar_canvas.cpp.

14.113.4.2 invTransform() `QwtPointPolar QwtPolarCanvas::invTransform ( const QPoint & pos ) const`

Translate a point from widget into plot coordinates

Parameters

pos	Point in widget coordinates of the plot canvas

Returns

Point in plot coordinates

See also

transform()

Definition at line 267 of file qwt_polar_canvas.cpp.

14.113.4.3 paintEvent() `void QwtPolarCanvas::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Paint event

Parameters

event	Paint event

Definition at line 186 of file qwt_polar_canvas.cpp.

14.113.4.4 plot() [1/2] QwtPolarPlot * QwtPolarCanvas::plot ( )

Returns

Parent plot widget

Definition at line 97 of file qwt_polar_canvas.cpp.

14.113.4.5 plot() [2/2] const QwtPolarPlot * QwtPolarCanvas::plot ( ) const

Returns

Parent plot widget

Definition at line 103 of file qwt_polar_canvas.cpp.

14.113.4.6 resizeEvent() void QwtPolarCanvas::resizeEvent (QResizeEvent * *event*) [override], [protected], [virtual]

Resize event

Parameters

event	Resize event

Definition at line 251 of file qwt_polar_canvas.cpp.

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

Changing the paint attributes.

Parameters

attribute	Paint attribute
on	On/Off

The default setting enables BackingStore

See also

testPaintAttribute(), paintCache()

Definition at line 118 of file qwt_polar_canvas.cpp.

14.113.4.8 testPaintAttribute() `bool QwtPolarCanvas::testPaintAttribute ( PaintAttribute attribute ) const`

Test whether a paint attribute is enabled

Parameters

attribute	Paint attribute

Returns

true if the attribute is enabled

See also

setPaintAttribute()

Definition at line 164 of file qwt_polar_canvas.cpp.

14.113.4.9 transform() `QPoint QwtPolarCanvas::transform ( const QwtPointPolar & polarPos ) const`

Translate a point from plot into widget coordinates

Parameters

polarPos	Point in plot coordinates

Returns

Point in widget coordinates

See also

transform()

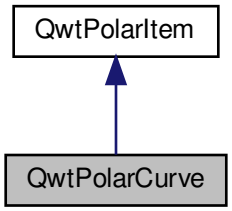
Definition at line 310 of file qwt_polar_canvas.cpp.

14.114 QwtPolarCurve Class Reference

An item, that represents a series of points.

`#include <qwt_polar_curve.h>`

Inheritance diagram for QwtPolarCurve:



Public Types

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

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

Public Member Functions

- QwtPolarCurve ()

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

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

  Test if a legend attribute is enabled.
- void setData (QwtSeriesData< QwtPointPolar > *data)
- const QwtSeriesData< QwtPointPolar > * data () const
- size_t dataSize () const
- QwtPointPolar sample (int i) const
- void setPen (const QPen &)

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

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

  Insert a curve fitter.
- QwtCurveFitter * curveFitter () const

- virtual void draw (QPainter *p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const override
- virtual void draw (QPainter *p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, int from, int to) const

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

Protected Member Functions

- void init ()

  Initialize data members.
- virtual void drawCurve (QPainter *, int style, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, int from, int to) const
- virtual void drawSymbols (QPainter *, const QwtSymbol &, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, int from, int to) const
- void drawLines (QPainter *, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, int from, int to) const

14.114.1 Detailed Description

An item, that represents a series of points.

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

See also

QwtPolarPlot, QwtSymbol, QwtScaleMap

Definition at line 30 of file qwt_polar_curve.h.

14.114.2 Member Typedef Documentation

14.114.2.1 LegendAttributes `typedef QFlags<LegendAttribute > QwtPolarCurve::LegendAttributes`

An ORed combination of LegendAttribute values.

Definition at line 75 of file qwt_polar_curve.h.

14.114.3 Member Enumeration Documentation

14.114.3.1 CurveStyle `enum QwtPolarCurve::CurveStyle`

Curve styles.

See also

setStyle(), style()

Enumerator

NoCurve	Don't draw a curve. Note: This doesn't affect the symbols.
Lines	Connect the points with straight lines. The lines might be interpolated depending on the 'Fitted' attribute. Curve fitting can be configured using setCurveFitter().
UserCurve	Values > 100 are reserved for user specific curve styles.

Definition at line 37 of file qwt_polar_curve.h.

14.114.3.2 LegendAttribute `enum QwtPolarCurve::LegendAttribute`

Attributes how to represent the curve on the legend.

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

See also

setLegendAttribute(), testLegendAttribute()

Enumerator

LegendShowLine	If the curveStyle() is not NoCurve a line is painted with the curvePen().
LegendShowSymbol	If the curve has a valid symbol it is painted.

Definition at line 63 of file qwt_polar_curve.h.

14.114.4 Constructor & Destructor Documentation

14.114.4.1 QwtPolarCurve() [1/2] `QwtPolarCurve::QwtPolarCurve ( const QwtText & title ) [explicit]`

Constructor

Parameters

title	title of the curve

Definition at line 76 of file qwt_polar_curve.cpp.

14.114.4.2 QwtPolarCurve() [2/2] `QwtPolarCurve::QwtPolarCurve ( const QString & title ) [explicit]`

Constructor

Parameters

title	title of the curve

Definition at line 86 of file qwt_polar_curve.cpp.

14.114.5 Member Function Documentation

14.114.5.1 boundingInterval() `QwtInterval QwtPolarCurve::boundingInterval ( int scaleId ) const [override], [virtual]`

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

Parameters

scaleId	Scale index

Returns

bounding interval

See also

QwtData::boundingRect()

Reimplemented from QwtPolarItem.

Definition at line 585 of file qwt_polar_curve.cpp.

14.114.5.2 curveFitter() `QwtCurveFitter * QwtPolarCurve::curveFitter ( ) const`

Returns

The curve fitter

See also

setCurveFitter()

Definition at line 261 of file qwt_polar_curve.cpp.

14.114.5.3 data() const QwtSeriesData< QwtPointPolar > * QwtPolarCurve::data ( ) const [inline]

Returns

the the curve data

Definition at line 144 of file qwt_polar_curve.h.

14.114.5.4 dataSize() size_t QwtPolarCurve::dataSize ( ) const

Returns

Number of points

See also

setData()

Definition at line 499 of file qwt_polar_curve.cpp.

14.114.5.5 draw() [1/2] void QwtPolarCurve::draw (

```
QPainter * painter,
const QwtScaleMap & azimuthMap,
const QwtScaleMap & radialMap,
const QPointF & pole,
double radius,
const QRectF & canvasRect ) const [override], [virtual]
```

Draw the curve

Parameters

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

Implements QwtPolarItem.

Definition at line 276 of file qwt_polar_curve.cpp.

14.114.5.6 draw() [2/2]

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

Draw an interval of the curve.

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
from	index of the first point to be painted
to	index of the last point to be painted. If to $<$ 0 the curve will be painted to its last point.

See also

drawCurve(), drawSymbols(),

Definition at line 299 of file qwt_polar_curve.cpp.

14.114.5.7 drawCurve()

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

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

Parameters

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

See also

draw(), drawLines()

Definition at line 341 of file qwt_polar_curve.cpp.

14.114.5.8 drawLines()

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

Draw lines

Parameters

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

See also

draw(), drawLines(), setCurveFitter()

Definition at line 367 of file qwt_polar_curve.cpp.

14.114.5.9 drawSymbols()

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

Draw symbols

Parameters

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

See also

setSymbol(), draw(), drawCurve()

Definition at line 459 of file qwt_polar_curve.cpp.

14.114.5.10 legendIcon() `QwtGraphic QwtPolarCurve::legendIcon ( int index, const QSizeF & size ) const [override], [virtual]`

Returns

Icon representing the curve on the legend

Parameters

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

See also

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

Reimplemented from QwtPolarItem.

Definition at line 513 of file qwt_polar_curve.cpp.

14.114.5.11 pen() `const QPen & QwtPolarCurve::pen ( ) const`

Returns

Pen used to draw the lines

See also

setPen()

Definition at line 212 of file qwt_polar_curve.cpp.

14.114.5.12 rtti() `int QwtPolarCurve::rtti ( ) const [override], [virtual]`

Returns

QwtPolarCurve::Rtti_PolarCurve

Reimplemented from QwtPolarItem.

Definition at line 113 of file qwt_polar_curve.cpp.

14.114.5.13 sample() `QwtPointPolar QwtPolarCurve::sample ( int i ) const [inline]`

Parameters

i	index

Returns

point at position i

Definition at line 153 of file qwt_polar_curve.h.

14.114.5.14 setCurveFitter() `void QwtPolarCurve::setCurveFitter ( QwtCurveFitter * curveFitter )`

Insert a curve fitter.

Parameters

curveFitter	Curve fitter

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

See also

curveFitter()

Definition at line 246 of file qwt_polar_curve.cpp.

14.114.5.15 setData() `void QwtPolarCurve::setData ( QwtSeriesData< QwtPointPolar > * data )`

Initialize data with a pointer to QwtSeriesData<QwtPointPolar>.

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

Parameters

data	Data

Definition at line 225 of file qwt_polar_curve.cpp.

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

Specify an attribute how to draw the legend identifier

Parameters

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

Definition at line 125 of file qwt_polar_curve.cpp.

14.114.5.17 setPen() `void QwtPolarCurve::setPen ( const QPen & pen )`

Assign a pen.

Parameters

pen	New pen

See also

pen()

Definition at line 199 of file qwt_polar_curve.cpp.

14.114.5.18 setStyle() `void QwtPolarCurve::setStyle ( CurveStyle style )`

Set the curve's drawing style

Parameters

style	Curve style

See also

CurveStyle, style()

Definition at line 152 of file qwt_polar_curve.cpp.

14.114.5.19 setSymbol() `void QwtPolarCurve::setSymbol ( QwtSymbol * symbol )`

Assign a symbol.

Parameters

symbol	Symbol

See also

symbol()

Definition at line 175 of file qwt_polar_curve.cpp.

14.114.5.20 style() `QwtPolarCurve::CurveStyle QwtPolarCurve::style ( ) const`

Returns

Current style

See also

CurveStyle, setStyle()

Definition at line 165 of file qwt_polar_curve.cpp.

14.114.5.21 symbol() `const QwtSymbol * QwtPolarCurve::symbol ( ) const`

Returns

The current symbol

See also

setSymbol()

Definition at line 189 of file qwt_polar_curve.cpp.

14.114.5.22 testLegendAttribute() `bool QwtPolarCurve::testLegendAttribute ( LegendAttribute attribute ) const`

Test if a legend attribute is enabled.

Parameters

attribute	Legend attribute

Returns

True if attribute is enabled

See also

LegendAttribute, setLegendAttribute()

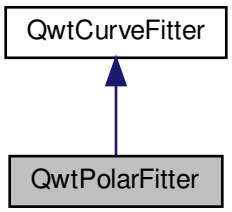
Definition at line 141 of file qwt_polar_curve.cpp.

14.115 QwtPolarFitter Class Reference

A simple curve fitter for polar points.

`#include <qwt_polar_fitter.h>`

Inheritance diagram for QwtPolarFitter:



Public Member Functions

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

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

Additional Inherited Members

14.115.1 Detailed Description

A simple curve fitter for polar points.

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

See also

QwtPolarCurve::setCurveFitter()

Definition at line 24 of file qwt_polar_fitter.h.

14.115.2 Constructor & Destructor Documentation

14.115.2.1 QwtPolarFitter() `QwtPolarFitter::QwtPolarFitter ( int stepCount = 5 )`

Constructor

Parameters

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

See also

setStepCount()

Definition at line 30 of file qwt_polar_fitter.cpp.

14.115.3 Member Function Documentation

14.115.3.1 fitCurve() `QPolygonF QwtPolarFitter::fitCurve ( const QPolygonF & points ) const [override], [virtual]`

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

Parameters

points	Array of points

Returns

Array of points including the additional points

Implements QwtCurveFitter.

Definition at line 72 of file qwt_polar_fitter.cpp.

14.115.3.2 fitCurvePath() `QPainterPath QwtPolarFitter::fitCurvePath ( const QPolygonF & points ) const [override], [virtual]`

Parameters

points	Series of data points

Returns

Curve path

See also

fitCurve()

Implements QwtCurveFitter.

Definition at line 110 of file qwt_polar_fitter.cpp.

14.115.3.3 setStepCount() `void QwtPolarFitter::setStepCount ( int stepCount )`

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

Parameters

stepCount	Number of steps

See also

stepCount()

Definition at line 51 of file qwt_polar_fitter.cpp.

14.115.3.4 stepCount() `int QwtPolarFitter::stepCount ( ) const`

Returns

Number of points, that will be inserted between 2 points

See also

setStepCount()

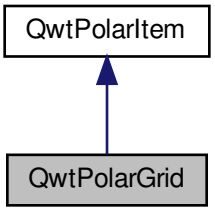
Definition at line 60 of file qwt_polar_fitter.cpp.

14.116 QwtPolarGrid Class Reference

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

`#include <qwt_polar_grid.h>`

Inheritance diagram for QwtPolarGrid:



Public Types

- enum DisplayFlag { SmartOriginLabel = 1 , HideMaxRadiusLabel = 2 , ClipAxisBackground = 4 , SmartScaleDraw = 8 , ClipGridLines = 16 }
- enum GridAttribute { AutoScaling = 0x01 }

  Grid attributes.
- typedef QFlags< DisplayFlag > DisplayFlags
- typedef QFlags< GridAttribute > GridAttributes

Public Member Functions

- QwtPolarGrid ()

  Constructor.
- virtual ~QwtPolarGrid ()

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

  Specify an attribute for the grid.
- bool testGridAttribute (GridAttribute) const
- void showGrid (int scaleId, bool show=true)
- bool isGridVisible (int scaleId) const
- void showMinorGrid (int scaleId, bool show=true)
- bool isMinorGridVisible (int scaleId) const
- void showAxis (int axisId, bool show=true)
- bool isAxisVisible (int axisId) const
- void setPen (const QPen &p)
- void setFont (const QFont &)

- void setMajorGridPen (const QPen &p)
- void setMajorGridPen (int scaleId, const QPen &p)
- QPen majorGridPen (int scaleId) const
- void setMinorGridPen (const QPen &p)
- void setMinorGridPen (int scaleId, const QPen &p)
- QPen minorGridPen (int scaleId) const
- void setAxisPen (int axisId, const QPen &p)
- QPen axisPen (int axisId) const
- void setAxisFont (int axisId, const QFont &p)
- QFont axisFont (int axisId) const
- void setScaleDraw (int axisId, QwtScaleDraw ∗)

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

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

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

Protected Member Functions

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

14.116.1 Detailed Description

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

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

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

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

See also

QwtPolarPlot, QwtPolar::Axis

Definition at line 41 of file qwt_polar_grid.h.

14.116.2 Member Typedef Documentation

14.116.2.1 DisplayFlags `typedef QFlags<DisplayFlag > QwtPolarGrid::DisplayFlags`

An ORed combination of DisplayFlag values.

Definition at line 88 of file qwt_polar_grid.h.

14.116.2.2 GridAttributes `typedef QFlags<GridAttribute > QwtPolarGrid::GridAttributes`

An ORed combination of GridAttribute values.

Definition at line 103 of file qwt_polar_grid.h.

14.116.3 Member Enumeration Documentation

14.116.3.1 DisplayFlag `enum QwtPolarGrid::DisplayFlag`

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

The default setting enables all flags.

See also

setDisplayFlag(), testDisplayFlag()

Enumerator

SmartOriginLabel	Try to avoid situations, where the label of the origin is painted over another axis.
HideMaxRadiusLabel	Often the outermost tick of the radial scale is close to the canvas border. With HideMaxRadiusLabel enabled it is not painted.
ClipAxisBackground	The tick labels of the radial scales might be hard to read, when they are painted on top of the radial grid lines (or on top of a curve/spectrogram). When ClipAxisBackground the bounding rect of each label is added to the clip region.
SmartScaleDraw	Don't paint the backbone of the radial axes, when they are very close to a line of the azimuth grid.
ClipGridLines	All grid lines are clipped against the plot area before being painted. When the plot is zoomed in this will have an significant impact on the performance of the painting code.

Definition at line 52 of file qwt_polar_grid.h.

14.116.3.2 GridAttribute `enum QwtPolarGrid::GridAttribute`

Grid attributes.

See also

setGridAttributes(), testGridAttributes()

Enumerator

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

Definition at line 94 of file qwt_polar_grid.h.

14.116.4 Constructor & Destructor Documentation

14.116.4.1 QwtPolarGrid() `QwtPolarGrid::QwtPolarGrid ( ) [explicit]`

Constructor.

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

Definition at line 84 of file qwt_polar_grid.cpp.

14.116.5 Member Function Documentation

14.116.5.1 axisFont() `QFont QwtPolarGrid::axisFont ( int axisId ) const`

Returns

Font for the tick labels of a specific axis

Parameters

axisId	Axis id (QwtPolar::Axis)

Definition at line 556 of file qwt_polar_grid.cpp.

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

Returns

Pen for painting a specific axis

Parameters

axisId	Axis id (QwtPolar::Axis)

See also

setAxisPen()

Definition at line 525 of file qwt_polar_grid.cpp.

14.116.5.3 azimuthScaleDraw() [1/2] QwtRoundScaleDraw * QwtPolarGrid::azimuthScaleDraw ( )

Returns

Scale draw for the azimuth scale

See also

setAzimuthScaleDraw(), scaleDraw()

Definition at line 1127 of file qwt_polar_grid.cpp.

14.116.5.4 azimuthScaleDraw() [2/2] const QwtRoundScaleDraw * QwtPolarGrid::azimuthScaleDraw ( ) const

Returns

Scale draw for the azimuth scale

See also

setAzimuthScaleDraw(), scaleDraw()

Definition at line 1117 of file qwt_polar_grid.cpp.

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

Draw the grid and axes

Parameters

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

Implements QwtPolarItem.

Definition at line 574 of file qwt_polar_grid.cpp.

14.116.5.6 drawAxis()

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

Paint an axis

Parameters

painter	Painter
axisId	Axis id (QwtPolar::Axis)

Definition at line 832 of file qwt_polar_grid.cpp.

14.116.5.7 drawCircles()

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

Draw circles

Parameters

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

Definition at line 751 of file qwt_polar_grid.cpp.

14.116.5.8 drawRays()

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

Draw lines from the pole

Parameters

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

Definition at line 681 of file qwt_polar_grid.cpp.

14.116.5.9 isAxisVisible()

```
bool QwtPolarGrid::isAxisVisible (
    int axisId ) const
```

Returns

true if the axis is visible

Parameters

axisId	Axis id (QwtPolar::Axis)

See also

showAxis()

Definition at line 343 of file qwt_polar_grid.cpp.

14.116.5.10 isGridVisible()

```
bool QwtPolarGrid::isGridVisible (
    int scaleId ) const
```

Returns

true if grid lines are enabled

Parameters

scale↩ Id	Scale id (QwtPolar::Scale)

See also

QwtPolar::Scale, showGrid()

Definition at line 272 of file qwt_polar_grid.cpp.

14.116.5.11 isMinorGridVisible() `bool QwtPolarGrid::isMinorGridVisible ( int scaleId ) const`

Returns

true if minor grid lines are enabled

Parameters

scale↩ Id	Scale id (QwtPolar::Scale)

See also

showMinorGrid()

Definition at line 308 of file qwt_polar_grid.cpp.

14.116.5.12 majorGridPen() `QPen QwtPolarGrid::majorGridPen ( int scaleId ) const`

Returns

Pen for painting the major grid lines of a specific scale

Parameters

scale↩ Id	Scale id (QwtPolar::Scale)

See also

setMajorGridPen(), minorGridPen()

Definition at line 454 of file qwt_polar_grid.cpp.

14.116.5.13 marginHint() `int QwtPolarGrid::marginHint ( ) const [override], [virtual]`

Returns

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

See also

QwtRoundScaleDraw::extent()

Reimplemented from QwtPolarItem.

Definition at line 1049 of file qwt_polar_grid.cpp.

14.116.5.14 minorGridPen() `QPen QwtPolarGrid::minorGridPen ( int scaleId ) const`

Returns

Pen for painting the minor grid lines of a specific scale

Parameters

scaleId	Scale id (QwtPolar::Scale)

Definition at line 510 of file qwt_polar_grid.cpp.

14.116.5.15 rtti() `int QwtPolarGrid::rtti ( ) const [override], [virtual]`

Returns

QwtPlotItem::Rtti_PolarGrid

Reimplemented from QwtPolarItem.

Definition at line 163 of file qwt_polar_grid.cpp.

14.116.5.16 scaleDraw() [1/2] `QwtScaleDraw * QwtPolarGrid::scaleDraw ( int axisId )`

Returns the scale draw of a specified axis

Parameters

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

Returns

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

See also

setScaleDraw(), azimuthScaleDraw()

Definition at line 1083 of file qwt_polar_grid.cpp.

14.116.5.17 scaleDraw() [2/2]

```
const QwtScaleDraw * QwtPolarGrid::scaleDraw (
    int axisId ) const
```

Returns the scale draw of a specified axis

Parameters

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

Returns

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

See also

azimuthScaleDraw()

Definition at line 1068 of file qwt_polar_grid.cpp.

14.116.5.18 setAxisFont()

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

Assign a font for the tick labels of a specific axis

Parameters

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

Definition at line 539 of file qwt_polar_grid.cpp.

14.116.5.19 setAxisPen() `void QwtPolarGrid::setAxisPen (`

```
int axisId,
const QPen & pen )
```

Assign a pen for painting an axis

Parameters

axisId	Axis id (QwtPolar::Axis)
pen	Pen

See also

axisPen()

Definition at line 235 of file qwt_polar_grid.cpp.

14.116.5.20 setAzimuthScaleDraw() `void QwtPolarGrid::setAzimuthScaleDraw (`

```
QwtRoundScaleDraw * scaleDraw )
```

Set a scale draw for the azimuth scale.

Parameters

scaleDraw	object responsible for drawing scales.

See also

azimuthScaleDraw(), setScaleDraw()

Definition at line 1139 of file qwt_polar_grid.cpp.

14.116.5.21 setDisplayFlag() `void QwtPolarGrid::setDisplayFlag (`

```
DisplayFlag flag,
bool on = true )
```

Change the display flags

Parameters

flag	See DisplayFlag
on	true/false

Definition at line 174 of file qwt_polar_grid.cpp.

14.116.5.22 setFont() `void QwtPolarGrid::setFont ( const QFont & font )`

Assign a font for all scale tick labels

Parameters

font	Font

See also

setAxisFont()

Definition at line 390 of file qwt_polar_grid.cpp.

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

Specify an attribute for the grid.

Parameters

attribute	Grid attribute
on	On/Off

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

Definition at line 205 of file qwt_polar_grid.cpp.

14.116.5.24 setMajorGridPen() [1/2] `void QwtPolarGrid::setMajorGridPen ( const QPen & pen )`

Assign a pen for the major grid lines

Parameters

pen	Pen

See also

setPen(), setMinorGridPen(), majorGridPen

Definition at line 412 of file qwt_polar_grid.cpp.

14.116.5.25 setMajorGridPen() [2/2]

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

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

Parameters

scaleId	Scale id (QwtPolar::Scale)
pen	Pen

See also

setPen(), setMinorGridPen(), majorGridPen

Definition at line 436 of file qwt_polar_grid.cpp.

14.116.5.26 setMinorGridPen() [1/2]

```
void QwtPolarGrid::setMinorGridPen (
    const QPen & pen )
```

Assign a pen for the minor grid lines

Parameters

pen	Pen

See also

setPen(), setMajorGridPen(), minorGridPen()

Definition at line 469 of file qwt_polar_grid.cpp.

14.116.5.27 setMinorGridPen() [2/2]

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

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

Parameters

scale↩ Id	Scale id (QwtPolar::Scale)
pen	Pen

See also

setPen(), setMajorGridPen(), minorGridPen

Definition at line 493 of file qwt_polar_grid.cpp.

14.116.5.28 setPen()

```
void QwtPolarGrid::setPen (
    const QPen & pen )
```

Assign a pen for all axes and grid lines

Parameters

pen	Pen

See also

setMajorGridPen(), setMinorGridPen(), setAxisPen()

Definition at line 357 of file qwt_polar_grid.cpp.

14.116.5.29 setScaleDraw()

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

Set a scale draw.

Parameters

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

See also

scaleDraw(), setAzimuthScaleDraw()

Definition at line 1099 of file qwt_polar_grid.cpp.

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

Show/Hide an axis

Parameters

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

See also

isAxisVisible()

Definition at line 324 of file qwt_polar_grid.cpp.

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

Show/Hide grid lines for a scale

Parameters

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

Definition at line 254 of file qwt_polar_grid.cpp.

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

Show/Hide minor grid lines for a scale

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

Parameters

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

See also

showGrid

Definition at line 290 of file qwt_polar_grid.cpp.

14.116.5.33 testDisplayFlag() `bool QwtPolarGrid::testDisplayFlag ( DisplayFlag flag ) const`

Returns

true, if flag is enabled

Parameters

flag	See DisplayFlag

Definition at line 191 of file qwt_polar_grid.cpp.

14.116.5.34 testGridAttribute() `bool QwtPolarGrid::testGridAttribute ( GridAttribute attribute ) const`

Returns

true, if attribute is enabled

See also

GridAttribute, setGridAttribute()

Definition at line 222 of file qwt_polar_grid.cpp.

14.116.5.35 updateScaleDiv() `void QwtPolarGrid::updateScaleDiv ( const QwtScaleDiv & azimuthScaleDiv, const QwtScaleDiv & radialScaleDiv, const QwtInterval & interval ) [override], [virtual]`

Update the item to changes of the axes scale division.

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

Parameters

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

See also

QwtPolarPlot::setGridAttributes()

Reimplemented from QwtPolarItem.

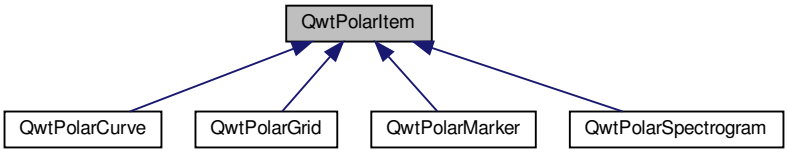
Definition at line 952 of file qwt_polar_grid.cpp.

14.117 QwtPolarItem Class Reference

Base class for items on a polar plot.

`#include <qwt_polar_item.h>`

Inheritance diagram for QwtPolarItem:



Public Types

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

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

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

  Render hints.
- typedef QFlags< ItemAttribute > ItemAttributes
- typedef QFlags< RenderHint > RenderHints

Public Member Functions

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

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

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

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

  Set the z value.
- void show ()

  Show the item.
- void hide ()

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

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

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

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

14.117.1 Detailed Description

Base class for items on a polar plot.

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

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

Definition at line 37 of file qwt_polar_item.h.

14.117.2 Member Typedef Documentation

14.117.2.1 ItemAttributes `typedef QFlags<ItemAttribute > QwtPolarItem::ItemAttributes`

An ORed combination of ItemAttribute values.

Definition at line 86 of file qwt_polar_item.h.

14.117.2.2 RenderHints `typedef QFlags<RenderHint > QwtPolarItem::RenderHints`

An ORed combination of RenderHint values.

Definition at line 98 of file qwt_polar_item.h.

14.117.3 Member Enumeration Documentation

14.117.3.1 ItemAttribute `enum QwtPolarItem::ItemAttribute`

Plot Item Attributes.

See also

setItemAttribute(), testItemAttribute()

Enumerator

Legend	The item is represented on the legend.
AutoScale	The boundingRect() of the item is included in the autoscaling calculation.

Definition at line 74 of file qwt_polar_item.h.

14.117.3.2 RenderHint `enum QwtPolarItem::RenderHint`

Render hints.

See also

setRenderHint(), testRenderHint()

Enumerator

RenderAntialiased	Enable antialiasing.

Definition at line 92 of file qwt_polar_item.h.

14.117.3.3 RttiValues `enum QwtPolarItem::RttiValues`

Runtime type information.

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

Enumerator

Rtti_PolarItem	Unspecific value, that can be used, when it doesn't matter.
Rtti_PolarGrid	For QwtPolarGrid.
Rtti_PolarMarker	For QwtPolarMarker.
Rtti_PolarCurve	For QwtPolarCurve.
Rtti_PolarSpectrogram	For QwtPolarSpectrogram.
Rtti_PolarUserItem	Values >= Rtti_PolarUserItem are reserved for plot items not implemented in the QwtPolar library.

Definition at line 46 of file qwt_polar_item.h.

14.117.4 Constructor & Destructor Documentation

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

Constructor

Parameters

title	Item title, f.e used on a legend

See also

setTitle()

Definition at line 48 of file qwt_polar_item.cpp.

14.117.5 Member Function Documentation

14.117.5.1 attach() void QwtPolarItem::attach (QwtPolarPlot * plot)

Attach the item to a plot.

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

Parameters

plot	Plot widget

See also

QwtPolarItem::detach()

Definition at line 74 of file qwt_polar_item.cpp.

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

Interval, that is necessary to display the item

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

Parameters

scaleId	Scale id (QwtPolar::Scale)

Returns

Bounding interval of the plot item for a specific scale

Reimplemented in QwtPolarSpectrogram, QwtPolarMarker, and QwtPolarCurve.

Definition at line 381 of file qwt_polar_item.cpp.

14.117.5.3 detach() void QwtPolarItem::detach ( )

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

detach() is equivalent to calling attach(NULL)

See also

attach()

Definition at line 95 of file qwt_polar_item.cpp.

14.117.5.4 draw()

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

Draw the item.

Parameters

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

Implemented in QwtPolarSpectrogram, QwtPolarMarker, QwtPolarGrid, and QwtPolarCurve.

14.117.5.5 isVisible()

```
bool QwtPolarItem::isVisible ( ) const
```

Returns

true if visible

See also

setVisible(), show(), hide()

Definition at line 344 of file qwt_polar_item.cpp.

14.117.5.6 itemChanged()

```
void QwtPolarItem::itemChanged ( )  [virtual]
```

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

See also

updateLegend()

Definition at line 355 of file qwt_polar_item.cpp.

14.117.5.7 legendChanged() `void QwtPolarItem::legendChanged ( ) [virtual]`

Update the legend of the parent plot.

See also

QwtPolarPlot::updateLegend(), itemChanged()

Definition at line 365 of file qwt_polar_item.cpp.

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

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

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

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

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

See also

title(), legendIcon(), QwtLegend

Definition at line 428 of file qwt_polar_item.cpp.

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

Returns

Icon representing the item on the legend

The default implementation returns an invalid icon

Parameters

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

See also

setLegendIconSize(), legendData()

Reimplemented in QwtPolarCurve.

Definition at line 462 of file qwt_polar_item.cpp.

14.117.5.10 legendIconSize() `QSize QwtPolarItem::legendIconSize ( ) const`

Returns

Legend icon size

See also

setLegendIconSize(), legendIcon()

Definition at line 308 of file qwt_polar_item.cpp.

14.117.5.11 marginHint() `int QwtPolarItem::marginHint ( ) const [virtual]`

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

Returns

Hint for the margin

Reimplemented in QwtPolarGrid.

Definition at line 478 of file qwt_polar_item.cpp.

14.117.5.12 plot() `QwtPolarPlot * QwtPolarItem::plot ( ) const`

Returns

Attached plot

Definition at line 118 of file qwt_polar_item.cpp.

14.117.5.13 renderThreadCount() `uint QwtPolarItem::renderThreadCount ( ) const`

Returns

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

Definition at line 282 of file qwt_polar_item.cpp.

14.117.5.14 rtti() `int QwtPolarItem::rtti ( ) const [virtual]`

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

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

Returns

rtti value

See also

RttiValues

Reimplemented in QwtPolarSpectrogram, QwtPolarMarker, QwtPolarGrid, and QwtPolarCurve.

Definition at line 112 of file qwt_polar_item.cpp.

14.117.5.15 setItemAttribute() `void QwtPolarItem::setItemAttribute ( ItemAttribute attribute, bool on = true )`

Toggle an item attribute

Parameters

attribute	Attribute type
on	true/false

See also

testItemAttribute(), ItemAttribute

Definition at line 201 of file qwt_polar_item.cpp.

14.117.5.16 setLegendIconSize() `void QwtPolarItem::setLegendIconSize ( const QSize & size )`

Set the size of the legend icon

The default setting is 8x8 pixels

Parameters

size	Size

See also

legendIconSize(), legendIcon()

Definition at line 295 of file qwt_polar_item.cpp.

14.117.5.17 setRenderHint()

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

Toggle an render hint

Parameters

hint	Render hint
on	true/false

See also

testRenderHint(), RenderHint

Definition at line 234 of file qwt_polar_item.cpp.

14.117.5.18 setRenderThreadCount()

```
void QwtPolarItem::setRenderThreadCount (
    uint numThreads )
```

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

The default setting is set to 1.

Parameters

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

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

Definition at line 272 of file qwt_polar_item.cpp.

14.117.5.19 setTitle() [1/2]

```
void QwtPolarItem::setTitle (
    const QString & title )
```

Set a new title

Parameters

title	Title

See also

title()

Definition at line 164 of file qwt_polar_item.cpp.

14.117.5.20 setTitle() [2/2] void QwtPolarItem::setTitle (const QwtText & *title*)

Set a new title

Parameters

title	Title

See also

title()

Definition at line 175 of file qwt_polar_item.cpp.

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

Show/Hide the item

Parameters

on	Show if true, otherwise hide

See also

isVisible(), show(), hide()

Definition at line 331 of file qwt_polar_item.cpp.

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

Set the z value.

Plot items are painted in increasing z-order.

Parameters

z	Z-value

See also

z(), QwtPolarItemDict::itemList()

Definition at line 142 of file qwt_polar_item.cpp.

14.117.5.23 testItemAttribute() `bool QwtPolarItem::testItemAttribute ( ItemAttribute attribute ) const`

Test an item attribute

Parameters

attribute	Attribute type

Returns

true/false

See also

setItemAttribute(), ItemAttribute

Definition at line 221 of file qwt_polar_item.cpp.

14.117.5.24 testRenderHint() `bool QwtPolarItem::testRenderHint ( RenderHint hint ) const`

Test a render hint

Parameters

hint	Render hint

Returns

true/false

See also

setRenderHint(), RenderHint

Definition at line 254 of file qwt_polar_item.cpp.

14.117.5.25 title() `const QwtText & QwtPolarItem::title ( ) const`

Returns

Title of the item

See also

setTitle()

Definition at line 188 of file qwt_polar_item.cpp.

14.117.5.26 updateScaleDiv() `void QwtPolarItem::updateScaleDiv (`

```
const QwtScaleDiv & azimuthScaleDiv,
const QwtScaleDiv & radialScaleDiv,
const QwtInterval & interval )  [virtual]
```

Update the item to changes of the axes scale division.

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

Parameters

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

See also

QwtPolarPlot::updateAxes()

Reimplemented in QwtPolarGrid.

Definition at line 403 of file qwt_polar_item.cpp.

14.117.5.27 z() `double QwtPolarItem::z ( ) const`

Plot items are painted in increasing z-order.

Returns

Z value

See also

setZ(), QwtPolarItemDict::itemList()

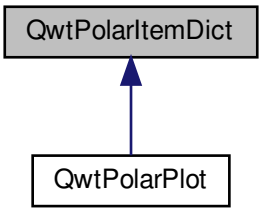
Definition at line 129 of file qwt_polar_item.cpp.

14.118 QwtPolarItemDict Class Reference

A dictionary for polar plot items.

`#include <qwt_polar_itemdict.h>`

Inheritance diagram for QwtPolarItemDict:



Public Member Functions

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

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

Protected Member Functions

- void insertItem (QwtPolarItem *)
- void removeItem (QwtPolarItem *)

14.118.1 Detailed Description

A dictionary for polar plot items.

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

See also

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

Definition at line 28 of file qwt_polar_itemdict.h.

14.118.2 Constructor & Destructor Documentation

14.118.2.1 QwtPolarItemDict() `QwtPolarItemDict::QwtPolarItemDict ( ) [explicit]`

Constructor

Auto deletion is enabled.

See also

setAutoDelete, attachItem

Definition at line 71 of file qwt_polar_itemdict.cpp.

14.118.2.2 ~QwtPolarItemDict() `QwtPolarItemDict::~QwtPolarItemDict ( )`

Destructor

If autoDelete is on, all attached items will be deleted

See also

setAutoDelete, autoDelete, attachItem

Definition at line 83 of file qwt_polar_itemdict.cpp.

14.118.3 Member Function Documentation

14.118.3.1 autoDelete() `bool QwtPolarItemDict::autoDelete ( ) const`

Returns

true if auto deletion is enabled

See also

setAutoDelete, attachItem

Definition at line 106 of file qwt_polar_itemdict.cpp.

14.118.3.2 detachItems() `void QwtPolarItemDict::detachItems (`

```
int rtti = QwtPolarItem::Rtti_PolarItem,
bool autoDelete = true )
```

Detach items from the dictionary

Parameters

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

Definition at line 140 of file qwt_polar_itemdict.cpp.

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

Insert a plot item

Parameters

item	PlotItem

See also

removeItem()

Definition at line 117 of file qwt_polar_itemdict.cpp.

14.118.3.4 itemList() `const QwtPolarItemList & QwtPolarItemDict::itemList ( ) const`

A QwtPolarItemList of all attached plot items.

Returns

List of all attached plot items.

Note

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

Definition at line 168 of file qwt_polar_itemdict.cpp.

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

Remove a plot item

Parameters

item	PlotItem

See also

insertItem()

Definition at line 128 of file qwt_polar_itemdict.cpp.

14.118.3.6 setAutoDelete() `void QwtPolarItemDict::setAutoDelete ( bool autoDelete )`

En/Disable Auto deletion

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

See also

autoDelete, attachItem

Definition at line 97 of file qwt_polar_itemdict.cpp.

14.119 QwtPolarLayout Class Reference

Layout class for QwtPolarPlot.

`#include <qwt_polar_layout.h>`

Public Types

- enum Option { IgnoreScrollbars = 0x01 , IgnoreFrames = 0x02 , IgnoreTitle = 0x04 , IgnoreLegend = 0x08 }
  Options to configure the plot layout engine.
- typedef QFlags< Option > Options

Public Member Functions

- QwtPolarLayout ()
  Constructor.
- virtual ~QwtPolarLayout ()
  Destructor.
- void setLegendPosition (QwtPolarPlot::LegendPosition pos, double ratio)
  Specify the position of the legend.
- void setLegendPosition (QwtPolarPlot::LegendPosition pos)
  Specify the position of the legend.
- QwtPolarPlot::LegendPosition legendPosition () const
- void setLegendRatio (double ratio)
- double legendRatio () const
- virtual void activate (const QwtPolarPlot *, const QRectF &rect, Options options=Options())
  Recalculate the geometry of all components.
- virtual void invalidate ()
- const QRectF & titleRect () const
- const QRectF & legendRect () const
- const QRectF & canvasRect () const

Protected Member Functions

- QRectF layoutLegend (Options options, QRectF &) const

14.119.1 Detailed Description

Layout class for QwtPolarPlot.

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

Definition at line 23 of file qwt_polar_layout.h.

14.119.2 Member Typedef Documentation

14.119.2.1 Options `typedef QFlags<Option > QwtPolarLayout::Options`

An ORed combination of Option values.

Definition at line 43 of file qwt_polar_layout.h.

14.119.3 Member Enumeration Documentation

14.119.3.1 Option `enum QwtPolarLayout::Option`

Options to configure the plot layout engine.

Enumerator

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

Definition at line 28 of file qwt_polar_layout.h.

14.119.4 Member Function Documentation

14.119.4.1 activate()

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

Recalculate the geometry of all components.

Parameters

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

See also

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

Definition at line 344 of file qwt_polar_layout.cpp.

14.119.4.2 canvasRect()

```
const QRectF & QwtPolarLayout::canvasRect ( ) const
```

Returns

Geometry for the canvas

See also

activate(), invalidate()

Definition at line 248 of file qwt_polar_layout.cpp.

14.119.4.3 invalidate()

```
void QwtPolarLayout::invalidate ( )  [virtual]
```

Invalidate the geometry of all components.

See also

activate()

Definition at line 257 of file qwt_polar_layout.cpp.

14.119.4.4 layoutLegend()

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

Find the geometry for the legend

Parameters

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

Returns

Geometry for the legend

Definition at line 269 of file qwt_polar_layout.cpp.

14.119.4.5 legendPosition() `QwtPolarPlot::LegendPosition QwtPolarLayout::legendPosition ( ) const`

Returns

Position of the legend

See also

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

Definition at line 196 of file qwt_polar_layout.cpp.

14.119.4.6 legendRatio() `double QwtPolarLayout::legendRatio ( ) const`

Returns

The relative size of the legend in the plot.

See also

setLegendPosition()

Definition at line 219 of file qwt_polar_layout.cpp.

14.119.4.7 legendRect() `const QRectF & QwtPolarLayout::legendRect ( ) const`

Returns

Geometry for the legend

See also

activate(), invalidate()

Definition at line 239 of file qwt_polar_layout.cpp.

14.119.4.8 setLegendPosition() [1/2] `void QwtPolarLayout::setLegendPosition ( QwtPolarPlot::LegendPosition pos )`

Specify the position of the legend.

Parameters

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

See also

QwtPolarPlot::setLegendPosition()

Definition at line 186 of file qwt_polar_layout.cpp.

14.119.4.9 setLegendPosition() [2/2]

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

Specify the position of the legend.

Parameters

pos	The legend's position.
ratio	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also

QwtPolarPlot::setLegendPosition()

Definition at line 141 of file qwt_polar_layout.cpp.

14.119.4.10 setLegendRatio()

```
void QwtPolarLayout::setLegendRatio (
    double ratio )
```

Specify the relative size of the legend in the plot

Parameters

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

Definition at line 210 of file qwt_polar_layout.cpp.

14.119.4.11 titleRect() `const QRectF & QwtPolarLayout::titleRect ( ) const`

Returns

Geometry for the title

See also

activate(), invalidate()

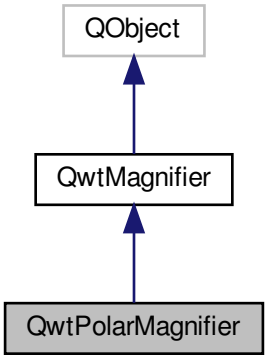
Definition at line 229 of file qwt_polar_layout.cpp.

14.120 QwtPolarMagnifier Class Reference

QwtPolarMagnifier provides zooming, by magnifying in steps.

`#include <qwt_polar_magnifier.h>`

Inheritance diagram for QwtPolarMagnifier:



Public Slots

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

  Unzoom the plot widget.

Public Member Functions

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

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

Protected Member Functions

- virtual void widgetKeyPressEvent (QKeyEvent *) override

14.120.1 Detailed Description

QwtPolarMagnifier provides zooming, by magnifying in steps.

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

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

See also

QwtPolarPanner, QwtPolarPlot, QwtPolarCanvas

Definition at line 30 of file qwt_polar_magnifier.h.

14.120.2 Constructor & Destructor Documentation

14.120.2.1 QwtPolarMagnifier() `QwtPolarMagnifier::QwtPolarMagnifier ( QwtPolarCanvas * canvas ) [explicit]`

Constructor

Parameters

canvas	Plot canvas to be magnified

Definition at line 34 of file qwt_polar_magnifier.cpp.

14.120.3 Member Function Documentation

14.120.3.1 canvas() [1/2] `QwtPolarCanvas * QwtPolarMagnifier::canvas ( )`

Returns

Observed plot canvas

Definition at line 74 of file qwt_polar_magnifier.cpp.

14.120.3.2 canvas() [2/2] const QwtPolarCanvas * QwtPolarMagnifier::canvas ( ) const

Returns

Observed plot canvas

Definition at line 80 of file qwt_polar_magnifier.cpp.

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

Returns

Key, and modifiers that are used for unzooming

Parameters

key	Key code
modifiers	Modifiers

See also

setUnzoomKey(), QwtPolarPlot::unzoom()

Definition at line 67 of file qwt_polar_magnifier.cpp.

14.120.3.4 plot() [1/2] QwtPolarPlot * QwtPolarMagnifier::plot ( )

Returns

Observed plot

Definition at line 86 of file qwt_polar_magnifier.cpp.

14.120.3.5 plot() [2/2] const QwtPolarPlot * QwtPolarMagnifier::plot ( ) const

Returns

observed plot

Definition at line 96 of file qwt_polar_magnifier.cpp.

14.120.3.6 rescale void QwtPolarMagnifier::rescale (double *factor*) [override], [virtual], [slot]

Zoom in/out the zoomed area

Parameters

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

Definition at line 129 of file qwt_polar_magnifier.cpp.

14.120.3.7 setUnzoomKey()

```
void QwtPolarMagnifier::setUnzoomKey (
    int key,
    int modifiers )
```

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

Parameters

key	Key code
modifiers	Modifiers

See also

getUnzoomKey(), QwtPolarPlot::unzoom()

Definition at line 54 of file qwt_polar_magnifier.cpp.

14.120.3.8 widgetKeyPressEvent()

```
void QwtPolarMagnifier::widgetKeyPressEvent (
    QKeyEvent * event )  [override], [protected], [virtual]
```

Handle a key press event for the observed widget.

Parameters

event	Key event

Reimplemented from QwtMagnifier.

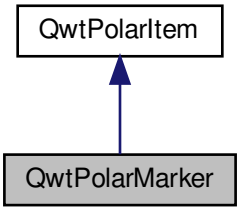
Definition at line 110 of file qwt_polar_magnifier.cpp.

14.121 QwtPolarMarker Class Reference

A class for drawing markers.

```
#include <qwt_polar_marker.h>
```

Inheritance diagram for QwtPolarMarker:



Public Member Functions

- QwtPolarMarker ()

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

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

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

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

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

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

Additional Inherited Members

14.121.1 Detailed Description

A class for drawing markers.

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

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

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

Definition at line 36 of file qwt_polar_marker.h.

14.121.2 Member Function Documentation

14.121.2.1 boundingInterval() QwtInterval QwtPolarMarker::boundingInterval (int *scaleId*) const [override], [virtual]

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

Parameters

scaleId	Scale index

Returns

bounding interval (== position)

See also

position()

Reimplemented from QwtPolarItem.

Definition at line 228 of file qwt_polar_marker.cpp.

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

Draw the marker

Parameters

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

Implements QwtPolarItem.

Definition at line 89 of file qwt_polar_marker.cpp.

14.121.2.3 label() `QwtText QwtPolarMarker::label ( ) const`

Returns

the label

See also

setLabel()

Definition at line 185 of file qwt_polar_marker.cpp.

14.121.2.4 labelAlignment() `Qt::Alignment QwtPolarMarker::labelAlignment ( ) const`

Returns

the label alignment

See also

setLabelAlignment()

Definition at line 214 of file qwt_polar_marker.cpp.

14.121.2.5 position() `QwtPointPolar QwtPolarMarker::position ( ) const`

Returns

Position of the marker

Definition at line 64 of file qwt_polar_marker.cpp.

14.121.2.6 rtti() `int QwtPolarMarker::rtti ( ) const [override], [virtual]`

Returns

QwtPolarItem::Rtti_PlotMarker

Reimplemented from QwtPolarItem.

Definition at line 58 of file qwt_polar_marker.cpp.

14.121.2.7 setLabel() `void QwtPolarMarker::setLabel ( const QwtText & label )`

Set the label.

Parameters

label	label text

See also

label()

Definition at line 172 of file qwt_polar_marker.cpp.

14.121.2.8 setLabelAlignment() `void QwtPolarMarker::setLabelAlignment ( Qt::Alignment align )`

Set the alignment of the label.

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

Parameters

align	Alignment. A combination of AlignTop, AlignBottom, AlignLeft, AlignRight, AlignCenter, AlgnHCenter, AlignVCenter.

See also

labelAlignment()

Definition at line 201 of file qwt_polar_marker.cpp.

14.121.2.9 setSymbol() `void QwtPolarMarker::setSymbol ( const QwtSymbol * symbol )`

Assign a symbol.

Parameters

symbol	New symbol

See also

symbol()

Definition at line 148 of file qwt_polar_marker.cpp.

14.121.2.10 symbol() `const QwtSymbol * QwtPolarMarker::symbol ( ) const`

Returns

the symbol

See also

setSymbol(), QwtSymbol

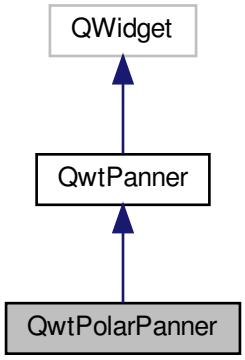
Definition at line 162 of file qwt_polar_marker.cpp.

14.122 QwtPolarPanner Class Reference

QwtPolarPanner provides panning of a polar plot canvas.

`#include <qwt_polar_panner.h>`

Inheritance diagram for QwtPolarPanner:



Public Slots

- virtual void movePlot (int dx, int dy)

Public Member Functions

- QwtPolarPanner (QwtPolarCanvas *)
  Create a plot panner for a polar plot canvas.
- virtual ~QwtPolarPanner ()
  Destructor.
- QwtPolarPlot * plot ()
- const QwtPolarPlot * plot () const
- QwtPolarCanvas * canvas ()
- const QwtPolarCanvas * canvas () const

Protected Member Functions

- virtual void widgetMousePressEvent (QMouseEvent ∗) override

Additional Inherited Members

14.122.1 Detailed Description

QwtPolarPanner provides panning of a polar plot canvas.

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

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

See also

QwtPolarMagnifier

Definition at line 30 of file qwt_polar_panner.h.

14.122.2 Member Function Documentation

14.122.2.1 canvas() [1/2] QwtPolarCanvas ∗ QwtPolarPanner::canvas ( )

Returns

observed plot canvas

Definition at line 29 of file qwt_polar_panner.cpp.

14.122.2.2 canvas() [2/2] const QwtPolarCanvas ∗ QwtPolarPanner::canvas ( ) const

Returns

observed plot canvas

Definition at line 35 of file qwt_polar_panner.cpp.

14.122.2.3 movePlot void QwtPolarPanner::movePlot (int *dx*, int *dy*) [virtual], [slot]

Adjust the zoomed area according to dx/dy

Parameters

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

See also

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

Definition at line 68 of file qwt_polar_panner.cpp.

14.122.2.4 plot() [1/2] QwtPolarPlot * QwtPolarPanner::plot ( )

Returns

observed plot

Definition at line 41 of file qwt_polar_panner.cpp.

14.122.2.5 plot() [2/2] const QwtPolarPlot * QwtPolarPanner::plot ( ) const

Returns

observed plot

Definition at line 51 of file qwt_polar_panner.cpp.

14.122.2.6 widgetMousePressEvent() void QwtPolarPanner::widgetMousePressEvent (QMouseEvent * *event*) [override], [protected], [virtual]

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

Parameters

event	Mouse event

Reimplemented from QwtPanner.

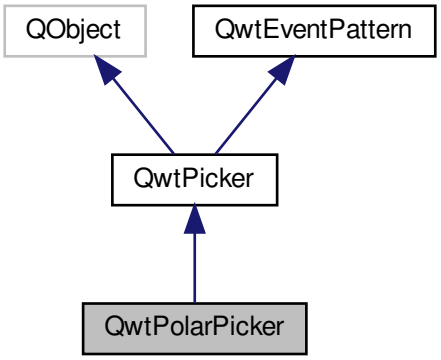
Definition at line 108 of file qwt_polar_panner.cpp.

14.123 QwtPolarPicker Class Reference

QwtPolarPicker provides selections on a plot canvas.

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

Inheritance diagram for QwtPolarPicker:



Signals

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

Public Member Functions

- QwtPolarPicker (QwtPolarCanvas *)

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

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

Protected Member Functions

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

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

Private Member Functions

- virtual QPainterPath pickArea () const override

Additional Inherited Members

14.123.1 Detailed Description

QwtPolarPicker provides selections on a plot canvas.

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

Definition at line 28 of file qwt_polar_picker.h.

14.123.2 Constructor & Destructor Documentation

14.123.2.1 QwtPolarPicker() [1/2] QwtPolarPicker::QwtPolarPicker (QwtPolarCanvas * *canvas*) [explicit]

Create a polar plot picker.

Parameters

canvas	Plot canvas to observe, also the parent object

Definition at line 25 of file qwt_polar_picker.cpp.

14.123.2.2 QwtPolarPicker() [2/2] QwtPolarPicker::QwtPolarPicker (RubberBand *rubberBand*, DisplayMode *trackerMode*, QwtPolarCanvas * *canvas*) [explicit]

Create a plot picker

Parameters

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

See also

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

Definition at line 43 of file qwt_polar_picker.cpp.

14.123.3 Member Function Documentation

14.123.3.1 append() `void QwtPolarPicker::append ( const QPoint & pos ) [override], [protected], [virtual]`

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

Parameters

pos	Additional point

See also

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

Note

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

Reimplemented from QwtPicker.

Definition at line 128 of file qwt_polar_picker.cpp.

14.123.3.2 appended `void QwtPolarPicker::appended ( const QwtPointPolar & pos ) [signal]`

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

Parameters

pos	Position of the appended point.

See also

append(). moved()

14.123.3.3 canvas() [1/2] `QwtPolarCanvas * QwtPolarPicker::canvas ( )`

Returns

Observed plot canvas

Definition at line 56 of file qwt_polar_picker.cpp.

14.123.3.4 canvas() [2/2] const QwtPolarCanvas * QwtPolarPicker::canvas ( ) const

Returns

Observed plot canvas

Definition at line 62 of file qwt_polar_picker.cpp.

14.123.3.5 end() bool QwtPolarPicker::end (bool *ok* = *true*) [override], [protected], [virtual]

Close a selection setting the state to inactive.

Parameters

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

Returns

true if the selection is accepted, false otherwise

Reimplemented from QwtPicker.

Definition at line 157 of file qwt_polar_picker.cpp.

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

Translate a point from widget into plot coordinates

Parameters

pos	Point in widget coordinates of the plot canvas

Returns

Point in plot coordinates

See also

transform(), canvas()

Definition at line 208 of file qwt_polar_picker.cpp.

14.123.3.7 move() `void QwtPolarPicker::move ( const QPoint & pos ) [override], [protected], [virtual]`

Move the last point of the selection

Parameters

pos	New position

See also

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

Note

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

Reimplemented from QwtPicker.

Definition at line 143 of file qwt_polar_picker.cpp.

14.123.3.8 moved `void QwtPolarPicker::moved ( const QwtPointPolar & pos ) [signal]`

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

Parameters

pos	Position of the moved last point of the selection.

See also

move(), appended()

14.123.3.9 pickArea() `QPainterPath QwtPolarPicker::pickArea ( ) const [override], [private], [virtual]`

Find the area of the observed widget, where selection might happen.

Returns

parentWidget()->contentsRect()

Reimplemented from QwtPicker.

Definition at line 229 of file qwt_polar_picker.cpp.

14.123.3.10 pickRect() `QRect QwtPolarPicker::pickRect ( ) const [virtual]`

Returns

Bounding rectangle of the region, where picking is supported.

Definition at line 221 of file qwt_polar_picker.cpp.

14.123.3.11 plot() `[1/2]` `QwtPolarPlot * QwtPolarPicker::plot ( )`

Returns

Plot widget, containing the observed plot canvas

Definition at line 68 of file qwt_polar_picker.cpp.

14.123.3.12 plot() `[2/2]` `const QwtPolarPlot * QwtPolarPicker::plot ( ) const`

Returns

Plot widget, containing the observed plot canvas

Definition at line 78 of file qwt_polar_picker.cpp.

14.123.3.13 selected `[1/2]` `void QwtPolarPicker::selected ( const QVector< QwtPointPolar > & points ) [signal]`

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

Parameters

points	Selected points

14.123.3.14 selected [2/2] void QwtPolarPicker::selected (const QwtPointPolar & *pos*) [signal]

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

Parameters

pos	Selected point

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

Translate a pixel position into a position string

Parameters

pos	Position in pixel coordinates

Returns

Position string

Reimplemented from QwtPicker.

Definition at line 93 of file qwt_polar_picker.cpp.

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

Translate a position into a position string.

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

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

Parameters

pos	Position

Returns

Position string

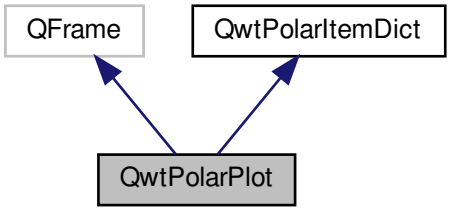
Definition at line 111 of file qwt_polar_picker.cpp.

14.124 QwtPolarPlot Class Reference

A plotting widget, displaying a polar coordinate system.

`#include <qwt_polar_plot.h>`

Inheritance diagram for QwtPolarPlot:



Public Types

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

Public Slots

- virtual void replot ()

  Redraw the plot.
- void autoRefresh ()

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

  Change the origin of the azimuth scale.

Signals

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

Public Member Functions

- QwtPolarPlot (QWidget *parent=NULL)
- QwtPolarPlot (const QwtText &title, QWidget *parent=NULL)
- virtual ~QwtPolarPlot ()

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

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

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

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

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

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

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

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

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

  Insert a legend.
- QwtAbstractLegend * legend ()
- const QwtAbstractLegend * legend () const
- void updateLegend ()
- void updateLegend (const QwtPolarItem *)

- QwtPolarLayout * plotLayout ()
- const QwtPolarLayout * plotLayout () const
- QwtInterval visibleInterval () const
- QRectF plotRect () const
- QRectF plotRect (const QRectF &) const

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

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

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

Protected Member Functions

- virtual bool event (QEvent *) override

  Qt event handler.
- virtual void resizeEvent (QResizeEvent *) override

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

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

14.124.1 Detailed Description

A plotting widget, displaying a polar coordinate system.

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

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

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

Definition at line 46 of file qwt_polar_plot.h.

14.124.2 Member Enumeration Documentation

14.124.2.1 LegendPosition `enum QwtPolarPlot::LegendPosition`

Position of the legend, relative to the canvas.

See also

insertLegend()

Enumerator

LeftLegend	The legend will be left from the canvas.
RightLegend	The legend will be right from the canvas.
BottomLegend	The legend will be below the canvas.
TopLegend	The legend will be between canvas and title.
ExternalLegend	External means that only the content of the legend will be handled by QwtPlot, but not its geometry. This might be interesting if an application wants to have a legend in an external window (or on the canvas). **Note** The legend is not painted by QwtPolarRenderer

Definition at line 59 of file qwt_polar_plot.h.

14.124.3 Constructor & Destructor Documentation

14.124.3.1 QwtPolarPlot() [1/2] QwtPolarPlot::QwtPolarPlot (QWidget * *parent* = *NULL*) [explicit]

Constructor

Parameters

parent	Parent widget

Definition at line 88 of file qwt_polar_plot.cpp.

14.124.3.2 QwtPolarPlot() [2/2] QwtPolarPlot::QwtPolarPlot (const QwtText & *title*, QWidget * *parent* = *NULL*)

Constructor

Parameters

title	Title text
parent	Parent widget

Definition at line 99 of file qwt_polar_plot.cpp.

14.124.4 Member Function Documentation

14.124.4.1 autoReplot() `bool QwtPolarPlot::autoReplot ( ) const`

Returns

true if the autoReplot option is set.

Definition at line 348 of file qwt_polar_plot.cpp.

14.124.4.2 azimuthOrigin() `double QwtPolarPlot::azimuthOrigin ( ) const`

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

Returns

Origin of the azimuth scale

See also

setAzimuthOrigin()

Definition at line 632 of file qwt_polar_plot.cpp.

14.124.4.3 canvas() [1/2] `QwtPolarCanvas * QwtPolarPlot::canvas ( )`

Returns

the plot's canvas

Definition at line 914 of file qwt_polar_plot.cpp.

14.124.4.4 canvas() [2/2] `const QwtPolarCanvas * QwtPolarPlot::canvas ( ) const`

Returns

the plot's canvas

Definition at line 920 of file qwt_polar_plot.cpp.

14.124.4.5 drawCanvas()

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

Redraw the canvas.

Parameters

painter	Painter used for drawing
canvasRect	Contents rect of the canvas

Definition at line 930 of file qwt_polar_plot.cpp.

14.124.4.6 drawItems()

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

Redraw the canvas items.

Parameters

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

Definition at line 975 of file qwt_polar_plot.cpp.

14.124.4.7 event()

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

Qt event handler.

Handles QEvent::LayoutRequest and QEvent::PolishRequest

Parameters

e	Qt Event

Returns

True, when the event was processed

Definition at line 762 of file qwt_polar_plot.cpp.

14.124.4.8 hasAutoScale() `bool QwtPolarPlot::hasAutoScale ( int scaleId ) const`

Returns

`true` if autoscaling is enabled

Parameters

scaleId	Scale index

See also

setAutoScale()

Definition at line 386 of file qwt_polar_plot.cpp.

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

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

The default implementation simply tries to unwrap a QwtPlotItem pointer:

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

Parameters

itemInfo	Plot item

Returns

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

See also

itemToInfo()

Definition at line 1357 of file qwt_polar_plot.cpp.

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

Insert a legend.

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

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

Parameters

legend	Legend
pos	The legend's position. For top/left position the number of columns will be limited to 1, otherwise it will be set to unlimited.
ratio	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also

legend(), QwtPolarLayout::legendPosition(), QwtPolarLayout::setLegendPosition()

Definition at line 191 of file qwt_polar_plot.cpp.

14.124.4.11 itemAttached

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

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

Parameters

plotItem	Plot item
on	Attached/Detached

14.124.4.12 itemToInfo()

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

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

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

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

Parameters

plotItem	Plot item

See also

infoToItem()

Definition at line 1337 of file qwt_polar_plot.cpp.

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

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

14.124.4.14 legend() `[1/2]` `QwtAbstractLegend * QwtPolarPlot::legend ( )`

Returns

the plot's legend

See also

insertLegend()

Definition at line 286 of file qwt_polar_plot.cpp.

14.124.4.15 legend() `[2/2]` `const QwtAbstractLegend * QwtPolarPlot::legend ( ) const`

Returns

the plot's legend

See also

insertLegend()

Definition at line 295 of file qwt_polar_plot.cpp.

14.124.4.16 legendDataChanged

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

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

Parameters

itemInfo	Info about a plot, build from itemToInfo()
data	Attributes of the entries (usually <= 1) for the plot item.

See also

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

14.124.4.17 plotBackground() `const QBrush & QwtPolarPlot::plotBackground ( ) const`

Returns

plot background brush

See also

plotBackground(), plotArea()

Definition at line 322 of file qwt_polar_plot.cpp.

14.124.4.18 plotLayout() [1/2] `QwtPolarLayout * QwtPolarPlot::plotLayout ( )`

Returns

Layout, responsible for the geometry of the plot components

Definition at line 1274 of file qwt_polar_plot.cpp.

14.124.4.19 plotLayout() [2/2] `const QwtPolarLayout * QwtPolarPlot::plotLayout ( ) const`

Returns

Layout, responsible for the geometry of the plot components

Definition at line 1282 of file qwt_polar_plot.cpp.

14.124.4.20 plotMarginHint() `int QwtPolarPlot::plotMarginHint ( ) const`

Returns

Maximum of all item margin hints.

See also

QwtPolarItem::marginHint()

Definition at line 1095 of file qwt_polar_plot.cpp.

14.124.4.21 plotRect() [1/2] QRectF QwtPolarPlot::plotRect ( ) const

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

Returns

Bounding rect of the plot area

Definition at line 1120 of file qwt_polar_plot.cpp.

14.124.4.22 plotRect() [2/2] QRectF QwtPolarPlot::plotRect (const QRectF & *canvasRect*) const

Calculate the bounding rect of the plot area.

The plot area depends on the zoom parameters.

Parameters

canvasRect	Rectangle of the canvas

Returns

Rectangle for displaying 100% of the plot

Definition at line 1133 of file qwt_polar_plot.cpp.

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

Redraw the plot.

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

See also

setAutoReplot()

Warning

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

Definition at line 899 of file qwt_polar_plot.cpp.

14.124.4.24 scaleDiv() [1/2] QwtScaleDiv * QwtPolarPlot::scaleDiv (int *scaleId*)

Return the scale division of a specified scale.

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

Parameters

scale↩ Id	Scale index

Returns

Scale division

See also

QwtScaleDiv, setScaleDiv(), setScale()

Definition at line 598 of file qwt_polar_plot.cpp.

14.124.4.25 scaleDiv() [2/2]

```
const QwtScaleDiv * QwtPolarPlot::scaleDiv (
    int scaleId ) const
```

Return the scale division of a specified scale.

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

Parameters

scale↩ Id	Scale index

Returns

Scale division

See also

QwtScaleDiv, setScaleDiv(), setScale()

Definition at line 579 of file qwt_polar_plot.cpp.

14.124.4.26 scaleEngine() [1/2]

```
QwtScaleEngine * QwtPolarPlot::scaleEngine (
    int scaleId )
```

Returns

Scale engine for a specific scale

Parameters

scale↩ Id	Scale index

See also

setScaleEngine()

Definition at line 499 of file qwt_polar_plot.cpp.

14.124.4.27 scaleEngine() [2/2]

```
const QwtScaleEngine * QwtPolarPlot::scaleEngine (
        int scaleId ) const
```

Returns

Scale engine for a specific scale

Parameters

scale↩ Id	Scale index

See also

setScaleEngine()

Definition at line 513 of file qwt_polar_plot.cpp.

14.124.4.28 scaleMap() [1/2]

```
QwtScaleMap QwtPolarPlot::scaleMap (
        int scaleId ) const
```

Build a scale map

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

Parameters

scale↩ Id	Scale index

Returns

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

See also

QwtScaleMap, transform(), invTransform()

Definition at line 710 of file qwt_polar_plot.cpp.

14.124.4.29 scaleMap() [2/2]

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

Build a scale map

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

Parameters

scaleId	Scale index
radius	Radius of the plot are in pixels

Returns

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

See also

QwtScaleMap, transform(), invTransform()

Definition at line 730 of file qwt_polar_plot.cpp.

14.124.4.30 scaleMaxMajor()

```
int QwtPolarPlot::scaleMaxMajor (
    int scaleId ) const
```

Returns

the maximum number of major ticks for a specified axis

Parameters

scaleId	Scale index

See also

setScaleMaxMajor()

Definition at line 460 of file qwt_polar_plot.cpp.

14.124.4.31 scaleMaxMinor() `int QwtPolarPlot::scaleMaxMinor ( int scaleId ) const`

Returns

the maximum number of minor ticks for a specified axis

Parameters

scaleId	Scale index

See also

setScaleMaxMinor()

Definition at line 423 of file qwt_polar_plot.cpp.

14.124.4.32 setAutoReplot() `void QwtPolarPlot::setAutoReplot ( bool enable = true )`

Set or reset the autoReplot option.

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

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

Parameters

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

See also

replot()

Definition at line 342 of file qwt_polar_plot.cpp.

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

Enable autoscaling.

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

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

Parameters

scaleId	Scale index

See also

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

Definition at line 368 of file qwt_polar_plot.cpp.

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

Change the origin of the azimuth scale.

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

Parameters

origin	New origin

See also

azimuthOrigin()

Definition at line 615 of file qwt_polar_plot.cpp.

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

Set the background of the plot area.

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

Parameters

brush	Background Brush

See also

plotBackground(), plotArea()

Definition at line 309 of file qwt_polar_plot.cpp.

14.124.4.36 setScale()

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

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

Parameters

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

See also

setScaleMaxMajor(), setAutoScale()

Definition at line 530 of file qwt_polar_plot.cpp.

14.124.4.37 setScaleDiv()

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

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

Parameters

scaleId	Scale index
scaleDiv	Scale division

See also

setScale(), setAutoScale()

Definition at line 554 of file qwt_polar_plot.cpp.

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

Change the scale engine for an axis

Parameters

scaleId	Scale index
scaleEngine	Scale engine

See also

axisScaleEngine()

Definition at line 476 of file qwt_polar_plot.cpp.

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

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

Parameters

scaleId	Scale index
maxMajor	maximum number of major steps

See also

scaleMaxMajor()

Definition at line 438 of file qwt_polar_plot.cpp.

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

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

Parameters

scaleId	Scale index
maxMinor	maximum number of minor steps

See also

scaleMaxMajor()

Definition at line 401 of file qwt_polar_plot.cpp.

14.124.4.41 setTitle() [1/2] void QwtPolarPlot::setTitle (const QString & *title*)

Change the plot's title

Parameters

title	New title

Definition at line 118 of file qwt_polar_plot.cpp.

14.124.4.42 setTitle() [2/2] void QwtPolarPlot::setTitle (const QwtText & *title*)

Change the plot's title

Parameters

title	New title

Definition at line 134 of file qwt_polar_plot.cpp.

14.124.4.43 title() QwtText QwtPolarPlot::title ( ) const

Returns

the plot's title

Definition at line 147 of file qwt_polar_plot.cpp.

14.124.4.44 titleLabel() [1/2] QwtTextLabel * QwtPolarPlot::titleLabel ( )

Returns

the plot's title

Definition at line 153 of file qwt_polar_plot.cpp.

14.124.4.45 titleLabel() [2/2] const QwtTextLabel * QwtPolarPlot::titleLabel ( ) const

Returns

the plot's title label.

Definition at line 159 of file qwt_polar_plot.cpp.

14.124.4.46 unzoom() void QwtPolarPlot::unzoom ( )

Unzoom the plot

See also

zoom()

Definition at line 668 of file qwt_polar_plot.cpp.

14.124.4.47 updateLegend() [1/2] void QwtPolarPlot::updateLegend ( )

Emit legendDataChanged() for all plot item

See also

QwtPlotItem::legendData(), legendDataChanged()

Definition at line 252 of file qwt_polar_plot.cpp.

14.124.4.48 updateLegend() [2/2] void QwtPolarPlot::updateLegend (const QwtPolarItem * *plotItem*)

Emit legendDataChanged() for a plot item

Parameters

plotItem	Plot item

See also

QwtPlotItem::legendData(), legendDataChanged()

Definition at line 268 of file qwt_polar_plot.cpp.

14.124.4.49 updateScale() `void QwtPolarPlot::updateScale ( int scaleId )`

Rebuild the scale

Parameters

scaleId	Scale index

Definition at line 1040 of file qwt_polar_plot.cpp.

14.124.4.50 visibleInterval() `QwtInterval QwtPolarPlot::visibleInterval ( ) const`

Returns

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

Definition at line 1170 of file qwt_polar_plot.cpp.

14.124.4.51 zoom() `void QwtPolarPlot::zoom ( const QwtPointPolar & zoomPos, double zoomFactor )`

Translate and in/decrease the zoom factor.

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

Setting an invalid zoom position disables zooming.

Parameters

zoomPos	Center of the translation
zoomFactor	Zoom factor

See also

unzoom(), zoomPos(), zoomFactor()

Definition at line 651 of file qwt_polar_plot.cpp.

14.124.4.52 zoomFactor() `double QwtPolarPlot::zoomFactor ( ) const`

Returns

Zoom factor

See also

zoom(), zoomPos()

Definition at line 691 of file qwt_polar_plot.cpp.

14.124.4.53 zoomPos() `QwtPointPolar QwtPolarPlot::zoomPos ( ) const`

Returns

Zoom position

See also

zoom(), zoomFactor()

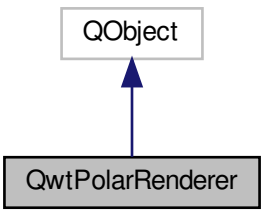
Definition at line 682 of file qwt_polar_plot.cpp.

14.125 QwtPolarRenderer Class Reference

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

`#include <qwt_polar_renderer.h>`

Inheritance diagram for QwtPolarRenderer:



Public Member Functions

- QwtPolarRenderer (QObject *parent=NULL)
- virtual ~QwtPolarRenderer ()

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

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

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

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

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

14.125.1 Detailed Description

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

Definition at line 35 of file qwt_polar_renderer.h.

14.125.2 Constructor & Destructor Documentation

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

Constructor

Parameters

parent	Parent object

Definition at line 90 of file qwt_polar_renderer.cpp.

14.125.3 Member Function Documentation

14.125.3.1 exportTo()

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

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

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

Parameters

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

See also

renderDocument()

Definition at line 430 of file qwt_polar_renderer.cpp.

14.125.3.2 render()

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

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

Parameters

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

Definition at line 326 of file qwt_polar_renderer.cpp.

14.125.3.3 renderDocument() [1/2]

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

Render a polar plot to a file

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

Parameters

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

Definition at line 113 of file qwt_polar_renderer.cpp.

14.125.3.4 renderDocument() [2/2]

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

Render a plot to a file

Supported formats are:

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

Parameters

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

See also

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

Definition at line 138 of file qwt_polar_renderer.cpp.

14.125.3.5 renderLegend()

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

Render the legend into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
rect	Bounding rectangle

Definition at line 411 of file qwt_polar_renderer.cpp.

14.125.3.6 renderTitle()

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

Render the title into a given rectangle.

Parameters

painter	Painter
rect	Bounding rectangle

Definition at line 391 of file qwt_polar_renderer.cpp.

14.125.3.7 renderTo() [1/2]

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

Render the plot to a `QPaintDevice`.

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

Parameters

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

See also

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

Definition at line 247 of file qwt_polar_renderer.cpp.

14.125.3.8 renderTo() [2/2] void QwtPolarRenderer::renderTo (QwtPolarPlot * *plot*, QPrinter & *printer*) const

Render the plot to a QPrinter.

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

Parameters

plot	Plot to be rendered
printer	Printer to paint on

See also

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

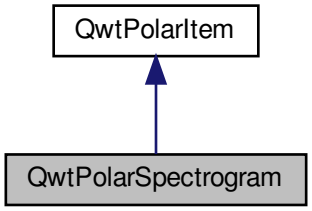
Definition at line 273 of file qwt_polar_renderer.cpp.

14.126 QwtPolarSpectrogram Class Reference

An item, which displays a spectrogram.

`#include <qwt_polar_spectrogram.h>`

Inheritance diagram for QwtPolarSpectrogram:



Public Types

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

Public Member Functions

- QwtPolarSpectrogram ()

  Constructor.
- virtual ~QwtPolarSpectrogram ()

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

Protected Member Functions

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

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

  Render a sub-rectangle of an image.

14.126.1 Detailed Description

An item, which displays a spectrogram.

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

See also

QwtRasterData, QwtColorMap

Definition at line 28 of file qwt_polar_spectrogram.h.

14.126.2 Member Typedef Documentation

14.126.2.1 PaintAttributes `typedef QFlags<PaintAttribute > QwtPolarSpectrogram::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 47 of file qwt_polar_spectrogram.h.

14.126.3 Member Enumeration Documentation

14.126.3.1 PaintAttribute `enum QwtPolarSpectrogram::PaintAttribute`

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

See also

setPaintAttribute(), testPaintAttribute()

Enumerator

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

Definition at line 37 of file qwt_polar_spectrogram.h.

14.126.4 Member Function Documentation

14.126.4.1 boundingInterval() `QwtInterval QwtPolarSpectrogram::boundingInterval ( int scaleId ) const [override], [virtual]`

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

Parameters

scaleId	Scale index

Returns

bounding interval (== position)

See also

position()

Reimplemented from QwtPolarItem.

Definition at line 443 of file qwt_polar_spectrogram.cpp.

14.126.4.2 colorMap() `const QwtColorMap * QwtPolarSpectrogram::colorMap ( ) const`

Returns

Color Map used for mapping the intensity values to colors

See also

setColorMap()

Definition at line 137 of file qwt_polar_spectrogram.cpp.

14.126.4.3 data() `const QwtRasterData * QwtPolarSpectrogram::data ( ) const`

Returns

Spectrogram data

See also

setData()

Definition at line 106 of file qwt_polar_spectrogram.cpp.

14.126.4.4 draw()

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

Draw the spectrogram

Parameters

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

Implements QwtPolarItem.

Definition at line 177 of file qwt_polar_spectrogram.cpp.

14.126.4.5 renderImage()

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

Render an image from the data and color map.

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

Parameters

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

Returns

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

See also

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

Definition at line 235 of file qwt_polar_spectrogram.cpp.

14.126.4.6 renderTile()

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

Render a sub-rectangle of an image.

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

Parameters

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

See also

setRenderThreadCount()

Note

renderTile needs to be reentrant

Definition at line 343 of file qwt_polar_spectrogram.cpp.

14.126.4.7 rtti() `int QwtPolarSpectrogram::rtti ( ) const [override], [virtual]`

Returns

QwtPolarItem::Rtti_PolarSpectrogram

Reimplemented from QwtPolarItem.

Definition at line 76 of file qwt_polar_spectrogram.cpp.

14.126.4.8 setColorMap() `void QwtPolarSpectrogram::setColorMap ( QwtColorMap * colorMap )`

Change the color map

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

Parameters

colorMap	Color Map

See also

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

Definition at line 122 of file qwt_polar_spectrogram.cpp.

14.126.4.9 setData() `void QwtPolarSpectrogram::setData ( QwtRasterData * data )`

Set the data to be displayed

Parameters

data	Spectrogram Data

See also

data()

Warning

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

Definition at line 91 of file qwt_polar_spectrogram.cpp.

14.126.4.10 setPaintAttribute()

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

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off

See also

testPaintAttribute()

Definition at line 149 of file qwt_polar_spectrogram.cpp.

14.126.4.11 testPaintAttribute()

```
bool QwtPolarSpectrogram::testPaintAttribute (
    PaintAttribute attribute ) const
```

Parameters

attribute	Paint attribute

Returns

True, when attribute has been set

See also

setPaintAttribute()

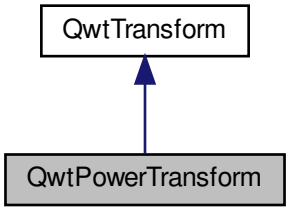
Definition at line 162 of file qwt_polar_spectrogram.cpp.

14.127 QwtPowerTransform Class Reference

A transformation using pow()

`#include <qwt_transform.h>`

Inheritance diagram for QwtPowerTransform:



Public Member Functions

- QwtPowerTransform (double exponent)
- virtual ~QwtPowerTransform ()

  Destructor.
- virtual double transform (double value) const override
- virtual double invTransform (double value) const override
- virtual QwtTransform * copy () const override

14.127.1 Detailed Description

A transformation using pow()

QwtPowerTransform preserves the sign of a value. F.e. a transformation with a factor of 2 transforms a value of -3 to -9 and v.v. Thus QwtPowerTransform can be used for scales including negative values.

Definition at line 125 of file qwt_transform.h.

14.127.2 Constructor & Destructor Documentation

14.127.2.1 QwtPowerTransform() `QwtPowerTransform::QwtPowerTransform ( double exponent ) [explicit]`

Constructor

Parameters

exponent	Exponent

Definition at line 121 of file qwt_transform.cpp.

14.127.3 Member Function Documentation

14.127.3.1 copy() `QwtTransform * QwtPowerTransform::copy ( ) const [override], [virtual]`

Returns

Clone of the transformation

Implements QwtTransform.

Definition at line 158 of file qwt_transform.cpp.

14.127.3.2 invTransform() `double QwtPowerTransform::invTransform ( double value ) const [override], [virtual]`

Parameters

value	Value to be transformed

Returns

Inverse exponentiation preserving the sign

Implements QwtTransform.

Definition at line 149 of file qwt_transform.cpp.

14.127.3.3 transform() `double QwtPowerTransform::transform ( double value ) const [override], [virtual]`

Parameters

value	Value to be transformed

Returns

Exponentiation preserving the sign

Implements QwtTransform.

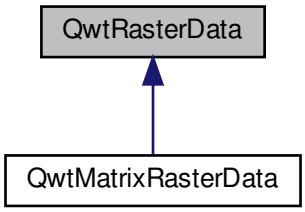
Definition at line 136 of file qwt_transform.cpp.

14.128 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 Attribute { WithoutGaps = 0x01 }
  Raster data attributes.
- enum ConrecFlag { IgnoreAllVerticesOnLevel = 0x01 , IgnoreOutOfRange = 0x02 }
  Flags to modify the contour algorithm.
- typedef QMap< double, QPolygonF > ContourLines
  Contour lines.
- typedef QFlags< Attribute > Attributes
- typedef QFlags< ConrecFlag > ConrecFlags

Public Member Functions

- QwtRasterData ()
  Constructor.
- virtual ~QwtRasterData ()
  Destructor.
- void setAttribute (Attribute, bool on=true)
- bool testAttribute (Attribute) const
- virtual QwtInterval interval (Qt::Axis) const =0
- virtual QRectF pixelHint (const QRectF &) const

Pixel hint.

- virtual void initRaster (const QRectF &, const QSize &raster)

  Initialize a raster.
- virtual void discardRaster ()

  Discard a raster.
- virtual double value (double x, double y) const =0
- virtual ContourLines contourLines (const QRectF &rect, const QSize &raster, const QList< double > &levels, ConrecFlags) const

14.128.1 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.

Gaps inside the bounding rectangle of the data can be indicated by NaN values (when WithoutGaps is disabled).

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).

QwtMatrixRasterData implements raster data, that returns values from a given 2D matrix.

See also

QwtMatrixRasterData

Definition at line 42 of file qwt_raster_data.h.

14.128.2 Member Typedef Documentation

14.128.2.1 Attributes `typedef QFlags<Attribute > QwtRasterData::Attributes`

An ORed combination of Attribute values.

Definition at line 74 of file qwt_raster_data.h.

14.128.2.2 ConrecFlags `typedef QFlags<ConrecFlag > QwtRasterData::ConrecFlags`

An ORed combination of ConrecFlag values.

Definition at line 86 of file qwt_raster_data.h.

14.128.3 Member Enumeration Documentation

14.128.3.1 Attribute `enum QwtRasterData::Attribute`

Raster data attributes.

Additional information that is used to improve processing of the data.

Enumerator

WithoutGaps	The bounding rectangle of the data is spanned by the interval(Qt::XAxis) and interval(Qt::YAxis). WithoutGaps indicates, that the data has no gaps (unknown values) in this area and the result of value() does not need to be checked for NaN values. Enabling this flag will have an positive effect on the performance of rendering a QwtPlotSpectrogram. The default setting is false. **Note** NaN values indicate an undefined value

Definition at line 54 of file qwt_raster_data.h.

14.128.3.2 ConrecFlag `enum QwtRasterData::ConrecFlag`

Flags to modify the contour algorithm.

Enumerator

IgnoreAllVerticesOnLevel	Ignore all vertices on the same level.
IgnoreOutOfRange	Ignore all values, that are out of range.

Definition at line 77 of file qwt_raster_data.h.

14.128.4 Member Function Documentation

14.128.4.1 contourLines()

```
QwtRasterData::ContourLines QwtRasterData::contourLines (
    const QRectF & rect,
    const QSize & raster,
    const QList< double > & levels,
    ConrecFlags flags ) const  [virtual]
```

Calculate contour lines

Parameters

rect	Bounding rectangle for the contour lines
raster	Number of data pixels of the raster data
levels	List of limits, where to insert contour lines
flags	Flags to customize the contouring algorithm

Returns

Calculated contour lines

An adaption of CONREC, a simple contouring algorithm. http://local.wasp.uwa.edu.↩au/~pbourke/papers/conrec/

Definition at line 286 of file qwt_raster_data.cpp.

14.128.4.2 discardRaster() `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 deleted now.

See also

initRaster(), value()

Definition at line 237 of file qwt_raster_data.cpp.

14.128.4.3 initRaster() `void QwtRasterData::initRaster ( const QRectF & area, 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 good idea to reimplement initRaster(), where the data is resampled and loaded into memory.

Parameters

area	Area of the raster
raster	Number of horizontal and vertical pixels

See also

initRaster(), value()

Definition at line 221 of file qwt_raster_data.cpp.

14.128.4.4 interval()

```
virtual QwtInterval QwtRasterData::interval (
    Qt::Axis ) const [pure virtual]
```

Returns

Bounding interval for an axis

See also

setInterval

Implemented in QwtMatrixRasterData.

14.128.4.5 pixelHint()

```
QRectF QwtRasterData::pixelHint (
    const QRectF & area ) const [virtual]
```

Pixel hint.

pixelHint() returns the geometry of a pixel, that can be used to calculate the resolution and alignment of the plot item, that is representing the data.

Width and height of the hint need to be the horizontal and vertical distances between 2 neighbored points. The center of the hint has to be the position of any point (it doesn't matter which one).

An empty hint indicates, that there are values for any detail level.

Limiting the resolution of the image might significantly improve the performance and heavily reduce the amount of memory when rendering a QImage from the raster data.

The default implementation returns an empty rectangle recommending to render in target device (f.e. screen) resolution.

Parameters

area	In most implementations the resolution of the data doesn't depend on the requested area.

Returns

Bounding rectangle of a pixel

Reimplemented in QwtMatrixRasterData.

Definition at line 267 of file qwt_raster_data.cpp.

14.128.4.6 setAttribute()

```
void QwtRasterData::setAttribute (
    Attribute attribute,
    bool on = true )
```

Specify an attribute of the data

Parameters

attribute	Attribute
on	On/Off /sa Attribute, testAttribute()

Definition at line 189 of file qwt_raster_data.cpp.

14.128.4.7 testAttribute() `bool QwtRasterData::testAttribute ( Attribute attribute ) const`

Returns

True, when attribute is enabled

See also

Attribute, setAttribute()

Definition at line 201 of file qwt_raster_data.cpp.

14.128.4.8 value() `virtual double QwtRasterData::value ( double x, double y ) const [pure virtual]`

Returns

the value at a raster position

Parameters

x	X value in plot coordinates
y	Y value in plot coordinates

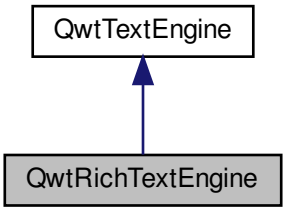
Implemented in QwtMatrixRasterData.

14.129 QwtRichTextEngine Class Reference

A text engine for Qt rich texts.

`#include <qwt_text_engine.h>`

Inheritance diagram for QwtRichTextEngine:



Public Member Functions

- QwtRichTextEngine ()

  Constructor.
- virtual double heightForWidth (const QFont &font, int flags, const QString &text, double width) const override
- virtual QSizeF textSize (const QFont &font, int flags, const QString &text) const override
- virtual void draw (QPainter *, const QRectF &rect, int flags, const QString &text) const override
- virtual bool mightRender (const QString &) const override
- virtual void textMargins (const QFont &, const QString &, double &left, double &right, double &top, double &bottom) const override

Additional Inherited Members

14.129.1 Detailed Description

A text engine for Qt rich texts.

QwtRichTextEngine renders Qt rich texts using the classes of the Scribe framework of Qt.

Definition at line 147 of file qwt_text_engine.h.

14.129.2 Member Function Documentation

14.129.2.1 draw()

```
void QwtRichTextEngine::draw (
        QPainter * painter,
        const QRectF & rect,
        int flags,
        const QString & text ) const  [override], [virtual]
```

Draw the text in a clipping rectangle

Parameters

painter	Painter
rect	Clipping rectangle
flags	Bitwise OR of the flags like in for QPainter::drawText()
text	Text to be rendered

Implements QwtTextEngine.

Definition at line 305 of file qwt_text_engine.cpp.

14.129.2.2 heightForWidth()

```
double QwtRichTextEngine::heightForWidth (
    const QFont & font,
    int flags,
    const QString & text,
    double width ) const  [override], [virtual]
```

Find the height for a given width

Parameters

font	Font of the text
flags	Bitwise OR of the flags used like in QPainter::drawText()
text	Text to be rendered
width	Width

Returns

Calculated height

Implements QwtTextEngine.

Definition at line 262 of file qwt_text_engine.cpp.

14.129.2.3 mightRender()

```
bool QwtRichTextEngine::mightRender (
    const QString & text ) const  [override], [virtual]
```

Test if a string can be rendered by this text engine

Parameters

text	Text to be tested

Returns

Qt::mightBeRichText(text);

Implements QwtTextEngine.

Definition at line 331 of file qwt_text_engine.cpp.

14.129.2.4 textMargins()

```
void QwtRichTextEngine::textMargins (
    const QFont & ,
    const QString & ,
    double & left,
    double & right,
    double & top,
    double & bottom ) const  [override], [virtual]
```

Return margins around the texts

Parameters

left	Return 0
right	Return 0
top	Return 0
bottom	Return 0

Implements QwtTextEngine.

Definition at line 344 of file qwt_text_engine.cpp.

14.129.2.5 textSize()

```
QSizeF QwtRichTextEngine::textSize (
    const QFont & font,
    int flags,
    const QString & text ) const  [override], [virtual]
```

Returns the size, that is needed to render text

Parameters

font	Font of the text
flags	Bitwise OR of the flags used like in QPainter::drawText()
text	Text to be rendered

Returns

Calculated size

Implements QwtTextEngine.

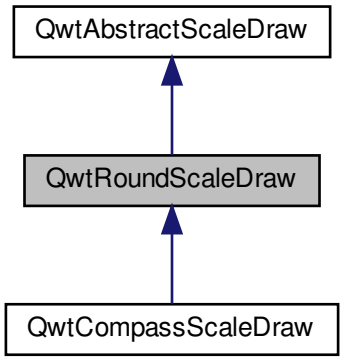
Definition at line 281 of file qwt_text_engine.cpp.

14.130 QwtRoundScaleDraw Class Reference

A class for drawing round scales.

`#include <qwt_round_scale_draw.h>`

Inheritance diagram for QwtRoundScaleDraw:



Public Member Functions

- QwtRoundScaleDraw ()
  Constructor.
- virtual ~QwtRoundScaleDraw ()
  Destructor.
- void setRadius (double radius)
- double radius () const
- void moveCenter (double x, double y)
  Move the center of the scale draw, leaving the radius unchanged.
- void moveCenter (const QPointF &)
- QPointF center () const
  Get the center of the scale.
- void setAngleRange (double angle1, double angle2)
  Adjust the baseline circle segment for round scales.
- virtual double extent (const QFont &) const override

Protected Member Functions

- virtual void drawTick (QPainter *, double value, double len) const override
- virtual void drawBackbone (QPainter *) const override
- virtual void drawLabel (QPainter *, double value) const override

Additional Inherited Members

14.130.1 Detailed Description

A class for drawing round scales.

QwtRoundScaleDraw can be used to draw round scales. The circle segment can be adjusted by setAngleRange(). The geometry of the scale can be specified with moveCenter() and 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.

Definition at line 31 of file qwt_round_scale_draw.h.

14.130.2 Constructor & Destructor Documentation

14.130.2.1 QwtRoundScaleDraw() `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].

Definition at line 44 of file qwt_round_scale_draw.cpp.

14.130.3 Member Function Documentation

14.130.3.1 drawBackbone() `void QwtRoundScaleDraw::drawBackbone ( QPainter * painter ) const [override], [protected], [virtual]`

Draws the baseline of the scale

Parameters

painter	Painter

See also

drawTick(), drawLabel()

Implements QwtAbstractScaleDraw.

Definition at line 224 of file qwt_round_scale_draw.cpp.

14.130.3.2 drawLabel()

```
void QwtRoundScaleDraw::drawLabel (
    QPainter * painter,
    double value ) const  [override], [protected], [virtual]
```

Draws the label for a major scale tick

Parameters

painter	Painter
value	Value

See also

drawTick(), drawBackbone()

Implements QwtAbstractScaleDraw.

Definition at line 145 of file qwt_round_scale_draw.cpp.

14.130.3.3 drawTick()

```
void QwtRoundScaleDraw::drawTick (
    QPainter * painter,
    double value,
    double len ) const  [override], [protected], [virtual]
```

Draw a tick

Parameters

painter	Painter
value	Value of the tick
len	Length of the tick

See also

drawBackbone(), drawLabel()

Implements QwtAbstractScaleDraw.

Definition at line 190 of file qwt_round_scale_draw.cpp.

14.130.3.4 extent()

```
double QwtRoundScaleDraw::extent (
    const QFont & font ) const  [override], [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

font	Font used for painting the labels

Returns

Calculated extent

See also

setMinimumExtent(), minimumExtent()

Warning

The implemented algorithm is not too smart and calculates only an upper limit, that might be a few pixels too large

Implements QwtAbstractScaleDraw.

Definition at line 255 of file qwt_round_scale_draw.cpp.

14.130.3.5 moveCenter() `void QwtRoundScaleDraw::moveCenter ( const QPointF & center )`

Move the center of the scale draw, leaving the radius unchanged

Parameters

center	New center

See also

setRadius()

Definition at line 90 of file qwt_round_scale_draw.cpp.

14.130.3.6 radius() `double QwtRoundScaleDraw::radius ( ) const`

Get the radius

Radius is the radius of the backbone without ticks and labels.

Returns

Radius of the scale

See also

setRadius(), extent()

Definition at line 79 of file qwt_round_scale_draw.cpp.

14.130.3.7 setAngleRange() `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

angle1	
angle2	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 or equal than 360 degrees above or below min(angle1, angle2), scale marks will not be drawn.
- If you need a counterclockwise scale, use QwtScaleDiv::setInterval()

Definition at line 118 of file qwt_round_scale_draw.cpp.

14.130.3.8 setRadius() `void QwtRoundScaleDraw::setRadius ( double radius )`

Change of radius the scale

Radius is the radius of the backbone without ticks and labels.

Parameters

radius	New Radius

See also

moveCenter()

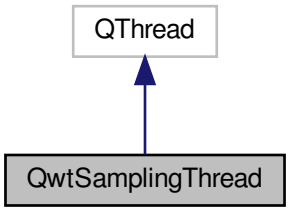
Definition at line 66 of file qwt_round_scale_draw.cpp.

14.131 QwtSamplingThread Class Reference

A thread collecting samples at regular intervals.

`#include <qwt_sampling_thread.h>`

Inheritance diagram for QwtSamplingThread:



Public Slots

- void setInterval (double interval)
- void stop ()

Public Member Functions

- virtual ~QwtSamplingThread ()

  Destructor.
- double interval () const
- double elapsed () const

Protected Member Functions

- QwtSamplingThread (QObject *parent=NULL)

  Constructor.
- virtual void run () override
- virtual void sample (double elapsed)=0

14.131.1 Detailed Description

A thread collecting samples at regular intervals.

Continuous signals are converted into a discrete signal by collecting samples at regular intervals. A discrete signal can be displayed by a QwtPlotSeriesItem on a QwtPlot widget.

QwtSamplingThread starts a thread calling periodically sample(), to collect and store (or emit) a single sample.

See also

QwtPlotCurve, QwtPlotSeriesItem

Definition at line 28 of file qwt_sampling_thread.h.

14.131.2 Member Function Documentation

14.131.2.1 elapsed() `double QwtSamplingThread::elapsed ( ) const`

Returns

Time (in ms) since the thread was started

See also

QThread::start(), run()

Definition at line 62 of file qwt_sampling_thread.cpp.

14.131.2.2 interval() `double QwtSamplingThread::interval ( ) const`

Returns

Interval (in ms), between 2 calls of sample()

See also

setInterval()

Definition at line 53 of file qwt_sampling_thread.cpp.

14.131.2.3 run() `void QwtSamplingThread::run ( ) [override], [protected], [virtual]`

Loop collecting samples started from QThread::start()

See also

stop()

Definition at line 83 of file qwt_sampling_thread.cpp.

14.131.2.4 sample() `virtual void QwtSamplingThread::sample ( double elapsed ) [protected], [pure virtual]`

Collect a sample

Parameters

elapsed	Time since the thread was started in seconds

Note

Due to a bug in previous version elapsed was passed as seconds instead of miliseconds. To avoid breaking existing code we stay with seconds for now.

14.131.2.5 setInterval `void QwtSamplingThread::setInterval ( double msecs ) [slot]`

Change the interval (in ms), when sample() is called. The default interval is 1000.0 (= 1s)

Parameters

msecs	Interval

See also

interval()

Definition at line 41 of file qwt_sampling_thread.cpp.

14.131.2.6 stop `void QwtSamplingThread::stop ( ) [slot]`

Terminate the collecting thread

See also

QThread::start(), run()

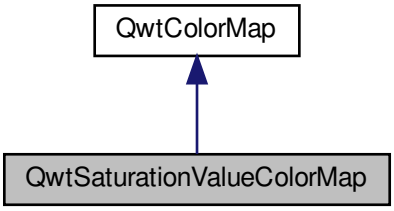
Definition at line 74 of file qwt_sampling_thread.cpp.

14.132 QwtSaturationValueColorMap Class Reference

QwtSaturationValueColorMap varies the saturation and/or value for a given hue in the HSV color model.

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

Inheritance diagram for QwtSaturationValueColorMap:



Public Member Functions

- QwtSaturationValueColorMap ()

  Constructor.
- virtual ~QwtSaturationValueColorMap ()

  Destructor.
- void setHue (int hue)

  Set the the hue coordinate.
- void setSaturationInterval (int sat1, int sat2)

  Set the interval for the saturation coordinate.
- void setValueInterval (int value1, int value2)

  Set the interval for the value coordinate.
- void setAlpha (int alpha)

  Set the the alpha coordinate.
- int hue () const
- int saturation1 () const
- int saturation2 () const
- int value1 () const
- int value2 () const
- int alpha () const
- virtual QRgb rgb (const QwtInterval &, double value) const override

Additional Inherited Members

14.132.1 Detailed Description

QwtSaturationValueColorMap varies the saturation and/or value for a given hue in the HSV color model.

Value and saturation are in the range of 0 to 255 while hue is in the range of 0 to 259.

See also

QwtHueColorMap

Definition at line 214 of file qwt_color_map.h.

14.132.2 Constructor & Destructor Documentation

14.132.2.1 QwtSaturationValueColorMap() `QwtSaturationValueColorMap::QwtSaturationValueColorMap ( )`

Constructor.

The value interval is initialized by 0 to 255, saturation by 255 to 255. Hue to 0 and alpha to 255.

So the default setting interpolates the value coordinate only.

See also

setHueInterval(), setSaturation(), setValue(), setValue()

Definition at line 972 of file qwt_color_map.cpp.

14.132.3 Member Function Documentation

14.132.3.1 alpha() `int QwtSaturationValueColorMap::alpha ( ) const`

Returns

Alpha coordinate

See also

setAlpha()

Definition at line 1127 of file qwt_color_map.cpp.

14.132.3.2 hue() `int QwtSaturationValueColorMap::hue ( ) const`

Returns

Hue coordinate

See also

setHue()

Definition at line 1082 of file qwt_color_map.cpp.

14.132.3.3 rgb()

```
QRgb QwtSaturationValueColorMap::rgb (
        const QwtInterval & interval,
        double value ) const  [override], [virtual]
```

Map a value of a given interval into a RGB value

Parameters

interval	Range for all values
value	Value to map into a RGB value

Returns

RGB value for value

Implements QwtColorMap.

Definition at line 1140 of file qwt_color_map.cpp.

14.132.3.4 saturation1() `int QwtSaturationValueColorMap::saturation1 ( ) const`

Returns

First saturation coordinate

See also

setSaturationInterval()

Definition at line 1091 of file qwt_color_map.cpp.

14.132.3.5 saturation2() `int QwtSaturationValueColorMap::saturation2 ( ) const`

Returns

Second saturation coordinate

See also

setSaturationInterval()

Definition at line 1100 of file qwt_color_map.cpp.

14.132.3.6 setAlpha() `void QwtSaturationValueColorMap::setAlpha ( int alpha )`

Set the the alpha coordinate.

alpha needs to be in the range 0 to 255, where 255 means opaque and 0 means transparent.

Parameters

alpha	Alpha coordinate

See also

alpha()

Definition at line 1067 of file qwt_color_map.cpp.

14.132.3.7 setHue() `void QwtSaturationValueColorMap::setHue ( int hue )`

Set the the hue coordinate.

Hue coordinates outside 0 to 359 will be interpreted as hue % 360..

Parameters

hue	Hue coordinate

See also

hue()

Definition at line 992 of file qwt_color_map.cpp.

14.132.3.8 setSaturationInterval() `void QwtSaturationValueColorMap::setSaturationInterval ( int saturation1, int saturation2 )`

Set the interval for the saturation coordinate.

When saturation1 == saturation2 the map interpolates between the value coordinates only

saturation1/saturation2 need to be in the range 0 to 255.

Parameters

saturation1	First saturation
saturation2	Second saturation

See also

saturation1(), saturation2(), setValueInterval()

Definition at line 1016 of file qwt_color_map.cpp.

14.132.3.9 setValueInterval() `void QwtSaturationValueColorMap::setValueInterval ( int value1, int value2 )`

Set the interval for the value coordinate.

When value1 == value2 the map interpolates between the saturation coordinates only.

value1/value2 need to be in the range 0 to 255.

Parameters

value1	First value
value2	Second value

See also

value1(), value2(), setSaturationInterval()

Definition at line 1043 of file qwt_color_map.cpp.

14.132.3.10 value1() `int QwtSaturationValueColorMap::value1 ( ) const`

Returns

First value coordinate

See also

setValueInterval()

Definition at line 1109 of file qwt_color_map.cpp.

14.132.3.11 value2() `int QwtSaturationValueColorMap::value2 ( ) const`

Returns

Second value coordinate

See also

setValueInterval()

Definition at line 1118 of file qwt_color_map.cpp.

14.133 QwtScaleArithmetic Class Reference

Arithmetic including a tolerance.

`#include <qwt_scale_engine.h>`

Static Public Member Functions

- static double ceilEps (double value, double intervalSize)
- static double floorEps (double value, double intervalSize)
- static double divideEps (double intervalSize, double numSteps)

  Divide an interval into steps.
- static double divideInterval (double intervalSize, int numSteps, uint base)

14.133.1 Detailed Description

Arithmetic including a tolerance.

Definition at line 22 of file qwt_scale_engine.h.

14.133.2 Member Function Documentation

14.133.2.1 ceilEps() `double QwtScaleArithmetic::ceilEps ( double value, double intervalSize ) [static]`

Ceil a value, relative to an interval

Parameters

value	Value to be ceiled
intervalSize	Interval size

Returns

Rounded value

See also

floorEps()

Definition at line 108 of file qwt_scale_engine.cpp.

14.133.2.2 divideEps() double QwtScaleArithmetic::divideEps (double *intervalSize*, double *numSteps*) [static]

Divide an interval into steps.

$stepSize = (intervalSize - intervalSize * 10e^{-6})/numSteps$

Parameters

intervalSize	Interval size
numSteps	Number of steps

Returns

Step size

Definition at line 143 of file qwt_scale_engine.cpp.

14.133.2.3 divideInterval() double QwtScaleArithmetic::divideInterval (double *intervalSize*, int *numSteps*, uint *base*) [static]

Calculate a step size for a given interval

Parameters

intervalSize	Interval size
numSteps	Number of steps
base	Base for the division (usually 10)

Returns

Calculated step size

Definition at line 160 of file qwt_scale_engine.cpp.

14.133.2.4 floorEps() double QwtScaleArithmetic::floorEps (double *value*, double *intervalSize*) [static]

Floor a value, relative to an interval

Parameters

value	Value to be floored
intervalSize	Interval size

Returns

Rounded value

See also

floorEps()

Definition at line 126 of file qwt_scale_engine.cpp.

14.134 QwtScaleDiv Class Reference

A class representing a scale division.

`#include <qwt_scale_div.h>`

Public Types

- enum TickType {
  NoTick = -1 , MinorTick , MediumTick , MajorTick ,
  NTickTypes }
  Scale tick types.

Public Member Functions

- QwtScaleDiv (double lowerBound=0.0, double upperBound=0.0)
- QwtScaleDiv (const QwtInterval &, QList< double >[NTickTypes])
- QwtScaleDiv (double lowerBound, double upperBound, QList< double >[NTickTypes])
- QwtScaleDiv (double lowerBound, double upperBound, const QList< double > &minorTicks, const QList< double > &mediumTicks, const QList< double > &majorTicks)
- bool operator== (const QwtScaleDiv &) const
  Equality operator.
- bool operator!= (const QwtScaleDiv &) const
  Inequality.
- void setInterval (double lowerBound, double upperBound)
- void setInterval (const QwtInterval &)
- QwtInterval interval () const
- void setLowerBound (double)
- double lowerBound () const
- void setUpperBound (double)
- double upperBound () const
- double range () const
- bool contains (double value) const
- void setTicks (int tickType, const QList< double > &)
- QList< double > ticks (int tickType) const
- bool isEmpty () const
  Check if the scale division is empty(lowerBound() == upperBound())
- bool isIncreasing () const
  Check if the scale division is increasing(lowerBound() <= upperBound())
- void invert ()
- QwtScaleDiv inverted () const
- QwtScaleDiv bounded (double lowerBound, double upperBound) const

14.134.1 Detailed Description

A class representing a scale division.

A Qwt scale is defined by its boundaries and 3 list for the positions of the major, medium and minor ticks.

The upperBound() might be smaller than the lowerBound() to indicate inverted scales.

Scale divisions can be calculated from a QwtScaleEngine.

See also

QwtScaleEngine::divideScale(), QwtPlot::setAxisScaleDiv(), QwtAbstractSlider::setScaleDiv()

Definition at line 33 of file qwt_scale_div.h.

14.134.2 Member Enumeration Documentation

14.134.2.1 TickType `enum QwtScaleDiv::TickType`

Scale tick types.

Enumerator

NoTick	No ticks.
MinorTick	Minor ticks.
MediumTick	Medium ticks.
MajorTick	Major ticks.
NTickTypes	Number of valid tick types.

Definition at line 37 of file qwt_scale_div.h.

14.134.3 Constructor & Destructor Documentation

14.134.3.1 QwtScaleDiv() [1/4]

```
QwtScaleDiv::QwtScaleDiv (
        double lowerBound = 0.0,
        double upperBound = 0.0 )  [explicit]
```

Construct a division without ticks

Parameters

lowerBound	First boundary
upperBound	Second boundary

Note

lowerBound might be greater than upperBound for inverted scales

Definition at line 21 of file qwt_scale_div.cpp.

14.134.3.2 QwtScaleDiv() [2/4]

```
QwtScaleDiv::QwtScaleDiv (
    const QwtInterval & interval,
    QList< double > ticks[NTickTypes] )  [explicit]
```

Construct a scale division

Parameters

interval	Interval
ticks	List of major, medium and minor ticks

Definition at line 33 of file qwt_scale_div.cpp.

14.134.3.3 QwtScaleDiv() [3/4]

```
QwtScaleDiv::QwtScaleDiv (
    double lowerBound,
    double upperBound,
    QList< double > ticks[NTickTypes] )  [explicit]
```

Construct a scale division

Parameters

lowerBound	First boundary
upperBound	Second boundary
ticks	List of major, medium and minor ticks

Note

lowerBound might be greater than upperBound for inverted scales

Definition at line 51 of file qwt_scale_div.cpp.

14.134.3.4 QwtScaleDiv() [4/4]

```
QwtScaleDiv::QwtScaleDiv (
    double lowerBound,
    double upperBound,
    const QList< double > & minorTicks,
    const QList< double > & mediumTicks,
    const QList< double > & majorTicks )  [explicit]
```

Construct a scale division

Parameters

lowerBound	First boundary
upperBound	Second boundary
minorTicks	List of minor ticks
mediumTicks	List medium ticks
majorTicks	List of major ticks

Note

lowerBound might be greater than upperBound for inverted scales

Definition at line 71 of file qwt_scale_div.cpp.

14.134.4 Member Function Documentation

14.134.4.1 bounded()

```
QwtScaleDiv QwtScaleDiv::bounded (
        double lowerBound,
        double upperBound ) const
```

Return a scale division with an interval [lowerBound, upperBound] where all ticks outside this interval are removed

Parameters

lowerBound	Lower bound
upperBound	Upper bound

Returns

Scale division with all ticks inside of the given interval

Note

lowerBound might be greater than upperBound for inverted scales

Definition at line 263 of file qwt_scale_div.cpp.

14.134.4.2 contains()

```
bool QwtScaleDiv::contains (
        double value ) const
```

Return if a value is between lowerBound() and upperBound()

Parameters

value	Value

Returns

true/false

Definition at line 212 of file qwt_scale_div.cpp.

14.134.4.3 interval() `QwtInterval QwtScaleDiv::interval ( ) const`

Returns

lowerBound -> upperBound

Definition at line 111 of file qwt_scale_div.cpp.

14.134.4.4 invert() `void QwtScaleDiv::invert ( )`

Invert the scale division

See also

inverted()

Definition at line 224 of file qwt_scale_div.cpp.

14.134.4.5 inverted() `QwtScaleDiv QwtScaleDiv::inverted ( ) const`

Returns

A scale division with inverted boundaries and ticks

See also

invert()

Definition at line 244 of file qwt_scale_div.cpp.

14.134.4.6 lowerBound() `double QwtScaleDiv::lowerBound ( ) const`

Returns

First boundary

See also

upperBound()

Definition at line 131 of file qwt_scale_div.cpp.

14.134.4.7 operator"!=() `bool QwtScaleDiv::operator!= ( const QwtScaleDiv & other ) const`

Inequality.

Returns

true if this instance is not equal to other

Definition at line 189 of file qwt_scale_div.cpp.

14.134.4.8 operator==() `bool QwtScaleDiv::operator== ( const QwtScaleDiv & other ) const`

Equality operator.

Returns

true if this instance is equal to other

Definition at line 168 of file qwt_scale_div.cpp.

14.134.4.9 range() `double QwtScaleDiv::range ( ) const`

Returns

upperBound() - lowerBound()

Definition at line 159 of file qwt_scale_div.cpp.

14.134.4.10 setInterval() [1/2] `void QwtScaleDiv::setInterval ( const QwtInterval & interval )`

Change the interval

Parameters

interval	Interval

Definition at line 102 of file qwt_scale_div.cpp.

14.134.4.11 setInterval() [2/2]

```
void QwtScaleDiv::setInterval (
    double lowerBound,
    double upperBound )
```

Change the interval

Parameters

lowerBound	First boundary
upperBound	Second boundary

Note

lowerBound might be greater than upperBound for inverted scales

Definition at line 91 of file qwt_scale_div.cpp.

14.134.4.12 setLowerBound()

```
void QwtScaleDiv::setLowerBound (
    double lowerBound )
```

Set the first boundary

Parameters

lowerBound	First boundary

See also

lowerBound(), setUpperBound()

Definition at line 122 of file qwt_scale_div.cpp.

14.134.4.13 setTicks()

```
void QwtScaleDiv::setTicks (
    int tickType,
    const QList< double > & ticks )
```

Assign ticks

Parameters

tickType	MinorTick, MediumTick or MajorTick
ticks	Values of the tick positions

Definition at line 297 of file qwt_scale_div.cpp.

14.134.4.14 setUpperBound() `void QwtScaleDiv::setUpperBound ( double upperBound )`

Set the second boundary

Parameters

upperBound	Second boundary

See also

upperBound(), setLowerBound()

Definition at line 142 of file qwt_scale_div.cpp.

14.134.4.15 ticks() `QList< double > QwtScaleDiv::ticks ( int tickType ) const`

Return a list of ticks

Parameters

tickType	MinorTick, MediumTick or MajorTick

Returns

Tick list

Definition at line 309 of file qwt_scale_div.cpp.

14.134.4.16 upperBound() `double QwtScaleDiv::upperBound ( ) const`

Returns

upper bound

See also

lowerBound()

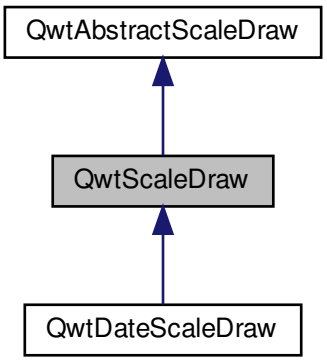
Definition at line 151 of file qwt_scale_div.cpp.

14.135 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 }

Public Member Functions

- QwtScaleDraw ()
  Constructor.
- virtual ~QwtScaleDraw ()
  Destructor.
- void getBorderDistHint (const QFont &, int &start, int &end) const
  Determine the minimum border distance.
- int minLabelDist (const QFont &) const
- int minLength (const QFont &) const
- virtual double extent (const QFont &) const override
- void move (double x, double y)
- void move (const QPointF &)
  Move the position of the scale.
- void setLength (double length)

- Alignment alignment () const
- void setAlignment (Alignment)
- Qt::Orientation orientation () const
- QPointF pos () const
- double length () const
- void setLabelAlignment (Qt::Alignment)

  Change the label flags.
- Qt::Alignment labelAlignment () const
- void setLabelRotation (double rotation)
- double labelRotation () const
- int maxLabelHeight (const QFont &) const
- int maxLabelWidth (const QFont &) const
- QPointF labelPosition (double value) const
- QRectF labelRect (const QFont &, double value) const
- QSizeF labelSize (const QFont &, double value) const
- QRect boundingLabelRect (const QFont &, double value) const

  Find the bounding rectangle for the label.

Protected Member Functions

- QTransform labelTransformation (const QPointF &, const QSizeF &) const
- virtual void drawTick (QPainter *, double value, double len) const override
- virtual void drawBackbone (QPainter *) const override
- virtual void drawLabel (QPainter *, double value) const override

14.135.1 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.

Definition at line 35 of file qwt_scale_draw.h.

14.135.2 Member Enumeration Documentation

14.135.2.1 Alignment `enum QwtScaleDraw::Alignment`

Alignment of the scale draw

See also

setAlignment(), alignment()

Enumerator

BottomScale	The scale is below.
TopScale	The scale is above.
LeftScale	The scale is left.
RightScale	The scale is right.

Definition at line 42 of file qwt_scale_draw.h.

14.135.3 Constructor & Destructor Documentation

14.135.3.1 QwtScaleDraw() `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.

Definition at line 292 of file qwt_scale_draw.cpp.

14.135.4 Member Function Documentation

14.135.4.1 alignment() `QwtScaleDraw::Alignment QwtScaleDraw::alignment ( ) const`

Return alignment of the scale

See also

setAlignment()

Returns

Alignment of the scale

Definition at line 309 of file qwt_scale_draw.cpp.

14.135.4.2 boundingLabelRect()

```
QRect QwtScaleDraw::boundingLabelRect (
    const QFont & font,
    double value ) const
```

Find the bounding rectangle for the label.

The coordinates of the rectangle are absolute (calculated from pos()). in direction of the tick.

Parameters

font	Font used for painting
value	Value

Returns

Bounding rectangle

See also

labelRect()

Definition at line 798 of file qwt_scale_draw.cpp.

14.135.4.3 drawBackbone()

```
void QwtScaleDraw::drawBackbone (
    QPainter * painter ) const  [override], [protected], [virtual]
```

Draws the baseline of the scale

Parameters

painter	Painter

See also

drawTick(), drawLabel()

Implements QwtAbstractScaleDraw.

Definition at line 668 of file qwt_scale_draw.cpp.

14.135.4.4 drawLabel()

```
void QwtScaleDraw::drawLabel (
    QPainter * painter,
    double value ) const  [override], [protected], [virtual]
```

Draws the label for a major scale tick

Parameters

painter	Painter
value	Value

See also

drawTick(), drawBackbone(), boundingLabelRect()

Implements QwtAbstractScaleDraw.

Definition at line 766 of file qwt_scale_draw.cpp.

14.135.4.5 drawTick()

```
void QwtScaleDraw::drawTick (
    QPainter * painter,
    double value,
    double len ) const  [override], [protected], [virtual]
```

Draw a tick

Parameters

painter	Painter
value	Value of the tick
len	Length of the tick

See also

drawBackbone(), drawLabel()

Implements QwtAbstractScaleDraw.

Definition at line 649 of file qwt_scale_draw.cpp.

14.135.4.6 extent()

```
double QwtScaleDraw::extent (
    const QFont & font ) const  [override], [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

font	Font used for painting the labels

Returns

Extent

See also

minLength()

Implements QwtAbstractScaleDraw.

Definition at line 523 of file qwt_scale_draw.cpp.

14.135.4.7 getBorderDistHint() `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

font	Font
start	Start border distance
end	End border distance

Definition at line 361 of file qwt_scale_draw.cpp.

14.135.4.8 labelAlignment() `Qt::Alignment QwtScaleDraw::labelAlignment ( ) const`

Returns

the label flags

See also

setLabelAlignment(), labelRotation()

Definition at line 982 of file qwt_scale_draw.cpp.

14.135.4.9 labelPosition() `QPointF QwtScaleDraw::labelPosition ( double value ) const`

Find the position, where to paint a label

The position has a distance that depends on the length of the ticks in direction of the alignment().

Parameters

value	Value

Returns

Position, where to paint a label

Definition at line 596 of file qwt_scale_draw.cpp.

14.135.4.10 labelRect() `QRectF QwtScaleDraw::labelRect ( const QFont & font, double value ) const`

Find the bounding rectangle for the label. The coordinates of the rectangle are relative to spacing + tick length from the backbone in direction of the tick.

Parameters

font	Font used for painting
value	Value

Returns

Bounding rectangle that is needed to draw a label

Definition at line 891 of file qwt_scale_draw.cpp.

14.135.4.11 labelRotation() `double QwtScaleDraw::labelRotation ( ) const`

Returns

the label rotation

See also

setLabelRotation(), labelAlignment()

Definition at line 943 of file qwt_scale_draw.cpp.

14.135.4.12 labelSize() `QSizeF QwtScaleDraw::labelSize ( const QFont & font, double value ) const`

Calculate the size that is needed to draw a label

Parameters

font	Label font
value	Value

Returns

Size that is needed to draw a label

Definition at line 916 of file qwt_scale_draw.cpp.

14.135.4.13 labelTransformation() `QTransform QwtScaleDraw::labelTransformation ( const QPointF & pos, const QSizeF & size ) const [protected]`

Calculate the transformation that is needed to paint a label depending on its alignment and rotation.

Parameters

pos	Position where to paint the label
size	Size of the label

Returns

Transformation matrix

See also

setLabelAlignment(), setLabelRotation()

Definition at line 821 of file qwt_scale_draw.cpp.

14.135.4.14 length() `double QwtScaleDraw::length ( ) const`

Returns

the length of the backbone

See also

setLength(), pos()

Definition at line 753 of file qwt_scale_draw.cpp.

14.135.4.15 maxLabelHeight() `int QwtScaleDraw::maxLabelHeight ( const QFont & font ) const`

Parameters

font	Font

Returns

the maximum height of a label

Definition at line 1014 of file qwt_scale_draw.cpp.

14.135.4.16 maxLabelWidth() `int QwtScaleDraw::maxLabelWidth ( const QFont & font ) const`

Parameters

font	Font

Returns

the maximum width of a label

Definition at line 991 of file qwt_scale_draw.cpp.

14.135.4.17 minLabelDist() `int QwtScaleDraw::minLabelDist ( const QFont & font ) const`

Determine the minimum distance between two labels, that is necessary that the texts don't overlap.

Parameters

font	Font

Returns

The maximum width of a label

See also

getBorderDistHint()

Definition at line 436 of file qwt_scale_draw.cpp.

14.135.4.18 minLength() `int QwtScaleDraw::minLength ( const QFont & font ) const`

Calculate the minimum length that is needed to draw the scale

Parameters

font	Font used for painting the labels

Returns

Minimum length that is needed to draw the scale

See also

extent()

Definition at line 560 of file qwt_scale_draw.cpp.

14.135.4.19 move() [1/2] void QwtScaleDraw::move (const QPointF & *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

pos	Origin of the scale

See also

pos(), setLength()

Definition at line 707 of file qwt_scale_draw.cpp.

14.135.4.20 move() [2/2] void QwtScaleDraw::move (double *x*, double *y*) [inline]

Move the position of the scale

Parameters

x	X coordinate
y	Y coordinate

See also

move(const QPointF &)

Definition at line 118 of file qwt_scale_draw.h.

14.135.4.21 orientation() `Qt::Orientation QwtScaleDraw::orientation ( ) const`

Return the orientation

TopScale, BottomScale are horizontal (Qt::Horizontal) scales, LeftScale, RightScale are vertical (Qt::Vertical) scales.

Returns

Orientation of the scale

See also

alignment()

Definition at line 337 of file qwt_scale_draw.cpp.

14.135.4.22 pos() `QPointF QwtScaleDraw::pos ( ) const`

Returns

Origin of the scale

See also

move(), length()

Definition at line 717 of file qwt_scale_draw.cpp.

14.135.4.23 setAlignment() `void QwtScaleDraw::setAlignment ( Alignment align )`

Set the alignment of the scale

Parameters

align	Alignment of the scale

The default alignment is QwtScaleDraw::BottomScale

See also

alignment()

Definition at line 322 of file qwt_scale_draw.cpp.

14.135.4.24 setLabelAlignment() `void QwtScaleDraw::setLabelAlignment ( Qt::Alignment alignment )`

Change the label flags.

Labels are aligned to the point tick length + spacing away from the backbone.

The alignment is relative to the orientation of the label text. In case of an flags of 0 the label will be aligned depending on the orientation of the scale:

QwtScaleDraw::TopScale: Qt::AlignHCenter | Qt::AlignTop
QwtScaleDraw::BottomScale: Qt::AlignHCenter | Qt::AlignBottom
QwtScaleDraw::LeftScale: Qt::AlignLeft | Qt::AlignVCenter
QwtScaleDraw::RightScale: Qt::AlignRight | Qt::AlignVCenter
Changing the alignment is often necessary for rotated labels.

Parameters

alignment	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().

Definition at line 973 of file qwt_scale_draw.cpp.

14.135.4.25 setLabelRotation() `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

rotation	Angle in degrees. When changing the label rotation, the label flags often needs to be adjusted too.

See also

setLabelAlignment(), labelRotation(), labelAlignment().

Definition at line 934 of file qwt_scale_draw.cpp.

14.135.4.26 setLength() `void QwtScaleDraw::setLength ( double length )`

Set the length of the backbone.

The length doesn't include the space needed for overlapping labels.

Parameters

length	Length of the backbone

See also

move(), minLabelDist()

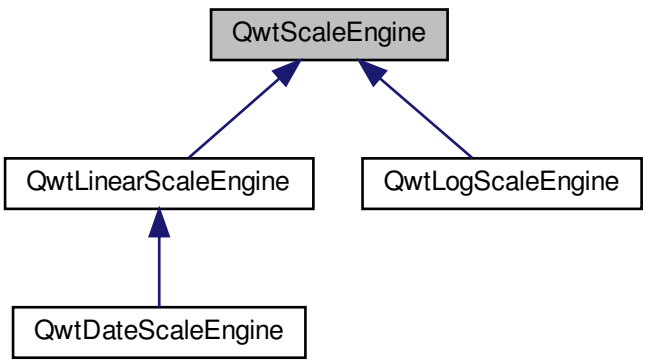
Definition at line 732 of file qwt_scale_draw.cpp.

14.136 QwtScaleEngine Class Reference

Base class for scale engines.

`#include <qwt_scale_engine.h>`

Inheritance diagram for QwtScaleEngine:



Public Types

- enum Attribute {
  NoAttribute = 0x00 , IncludeReference = 0x01 , Symmetric = 0x02 , Floating = 0x04 ,
  Inverted = 0x08 }
- typedef QFlags< Attribute > Attributes

Public Member Functions

- QwtScaleEngine (uint base=10)
- virtual ~QwtScaleEngine ()

  Destructor.
- void setBase (uint base)
- uint base () const
- void setAttribute (Attribute, bool on=true)
- bool testAttribute (Attribute) const
- void setAttributes (Attributes)
- Attributes attributes () const
- void setReference (double)

  Specify a reference point.
- double reference () const
- void setMargins (double lower, double upper)

  Specify margins at the scale's endpoints.
- double lowerMargin () const
- double upperMargin () const
- virtual void autoScale (int maxNumSteps, double &x1, double &x2, double &stepSize) const =0
- virtual QwtScaleDiv divideScale (double x1, double x2, int maxMajorSteps, int maxMinorSteps, double step↩
  Size=0.0) const =0

  Calculate a scale division.
- void setTransformation (QwtTransform *)
- QwtTransform * transformation () const

Protected Member Functions

- bool contains (const QwtInterval &, double value) const
- QList< double > strip (const QList< double > &, const QwtInterval &) const
- double divideInterval (double intervalSize, int numSteps) const
- QwtInterval buildInterval (double value) const

  Build an interval around a value.

14.136.1 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 and linear scales.

Definition at line 45 of file qwt_scale_engine.h.

14.136.2 Member Typedef Documentation

14.136.2.1 Attributes `typedef QFlags<Attribute > QwtScaleEngine::Attributes`

An ORed combination of Attribute values.

Definition at line 78 of file qwt_scale_engine.h.

14.136.3 Member Enumeration Documentation

14.136.3.1 Attribute `enum QwtScaleEngine::Attribute`

Layout attributes

See also

setAttribute(), testAttribute(), reference(), lowerMargin(), upperMargin()

Enumerator

NoAttribute	No attributes.
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.

Definition at line 54 of file qwt_scale_engine.h.

14.136.4 Constructor & Destructor Documentation

14.136.4.1 QwtScaleEngine() `QwtScaleEngine::QwtScaleEngine ( uint base = 10 ) [explicit]`

Constructor

Parameters

base	Base of the scale engine

See also

setBase()

Definition at line 222 of file qwt_scale_engine.cpp.

14.136.5 Member Function Documentation

14.136.5.1 attributes() `QwtScaleEngine::Attributes QwtScaleEngine::attributes ( ) const`

Returns

Scale attributes

See also

Attribute, setAttributes(), testAttribute()

Definition at line 451 of file qwt_scale_engine.cpp.

14.136.5.2 autoScale()

```
virtual void QwtScaleEngine::autoScale (
    int maxNumSteps,
    double & x1,
    double & x2,
    double & stepSize ) const  [pure virtual]
```

Align and divide an interval

Parameters

maxNumSteps	Max. number of steps
x1	First limit of the interval (In/Out)
x2	Second limit of the interval (In/Out)
stepSize	Step size (Return value)

Implemented in QwtLogScaleEngine, QwtLinearScaleEngine, and QwtDateScaleEngine.

14.136.5.3 base() `uint QwtScaleEngine::base ( ) const`

Returns

base Base of the scale engine

See also

setBase()

Definition at line 500 of file qwt_scale_engine.cpp.

14.136.5.4 buildInterval() `QwtInterval QwtScaleEngine::buildInterval ( double value ) const [protected]`

Build an interval around a value.

In case of v == 0.0 the interval is [-0.5, 0.5], otherwise it is [0.5 ∗ v, 1.5 ∗ v]

Parameters

value	Initial value

Returns

Calculated interval

Definition at line 395 of file qwt_scale_engine.cpp.

14.136.5.5 contains() `bool QwtScaleEngine::contains ( const QwtInterval & interval, double value ) const [protected]`

Check if an interval "contains" a value

Parameters

interval	Interval
value	Value

Returns

True, when the value is inside the interval

Definition at line 341 of file qwt_scale_engine.cpp.

14.136.5.6 divideInterval() `double QwtScaleEngine::divideInterval ( double intervalSize, int numSteps ) const [protected]`

Calculate a step size for an interval size

Parameters

intervalSize	Interval size
numSteps	Number of steps

Returns

Step size

Definition at line 326 of file qwt_scale_engine.cpp.

14.136.5.7 divideScale()

```
virtual QwtScaleDiv QwtScaleEngine::divideScale (
    double x1,
    double x2,
    int maxMajorSteps,
    int maxMinorSteps,
    double stepSize = 0.0 ) const  [pure virtual]
```

Calculate a scale division.

Parameters

x1	First interval limit
x2	Second interval limit
maxMajorSteps	Maximum for the number of major steps
maxMinorSteps	Maximum number of minor steps
stepSize	Step size. If stepSize == 0.0, the scaleEngine calculates one.

Returns

Calculated scale division

Implemented in QwtLogScaleEngine, QwtLinearScaleEngine, and QwtDateScaleEngine.

14.136.5.8 lowerMargin()

```
double QwtScaleEngine::lowerMargin ( ) const
```

Returns

the margin at the lower end of the scale The default margin is 0.

See also

setMargins()

Definition at line 280 of file qwt_scale_engine.cpp.

14.136.5.9 reference() `double QwtScaleEngine::reference ( ) const`

Returns

the reference value

See also

setReference(), setAttribute()

Definition at line 474 of file qwt_scale_engine.cpp.

14.136.5.10 setAttribute() `void QwtScaleEngine::setAttribute (`

```
Attribute attribute,
bool on = true )
```

Change a scale attribute

Parameters

attribute	Attribute to change
on	On/Off

See also

Attribute, testAttribute()

Definition at line 417 of file qwt_scale_engine.cpp.

14.136.5.11 setAttributes() `void QwtScaleEngine::setAttributes (`

```
Attributes attributes )
```

Change the scale attribute

Parameters

attributes	Set scale attributes

See also

Attribute, attributes()

Definition at line 442 of file qwt_scale_engine.cpp.

14.136.5.12 setBase() void QwtScaleEngine::setBase (uint *base*)

Set the base of the scale engine

While a base of 10 is what 99.9% of all applications need certain scales might need a different base: f.e 2

The default setting is 10

Parameters

base	Base of the engine

See also

base()

Definition at line 491 of file qwt_scale_engine.cpp.

14.136.5.13 setMargins() void QwtScaleEngine::setMargins (double *lower*, double *upper*)

Specify margins at the scale's endpoints.

Parameters

lower	minimum distance between the scale's lower boundary and the smallest enclosed value
upper	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

- QwtLogScaleEngine measures the margins in decades.

See also

upperMargin(), lowerMargin()

Definition at line 312 of file qwt_scale_engine.cpp.

14.136.5.14 setReference() void QwtScaleEngine::setReference (double *reference*)

Specify a reference point.

Parameters

reference	New reference value

The reference point is needed if options IncludeReference or Symmetric are active. Its default value is 0.0.

See also

Attribute

Definition at line 465 of file qwt_scale_engine.cpp.

14.136.5.15 setTransformation()

```
void QwtScaleEngine::setTransformation (
    QwtTransform * transform )
```

Assign a transformation

Parameters

transform	Transformation

The transformation object is used as factory for clones that are returned by transformation()

The scale engine takes ownership of the transformation.

See also

QwtTransform::copy(), transformation()

Definition at line 248 of file qwt_scale_engine.cpp.

14.136.5.16 strip()

```
QList< double > QwtScaleEngine::strip (
    const QList< double > & ticks,
    const QwtInterval & interval ) const  [protected]
```

Remove ticks from a list, that are not inside an interval

Parameters

ticks	Tick list
interval	Interval

Returns

Stripped tick list

Definition at line 364 of file qwt_scale_engine.cpp.

14.136.5.17 testAttribute() `bool QwtScaleEngine::testAttribute ( Attribute attribute ) const`

Returns

True, if attribute is enabled.

Parameters

attribute	Attribute to be tested

See also

Attribute, setAttribute()

Definition at line 431 of file qwt_scale_engine.cpp.

14.136.5.18 transformation() `QwtTransform * QwtScaleEngine::transformation ( ) const`

Create and return a clone of the transformation of the engine. When the engine has no special transformation NULL is returned, indicating no transformation.

Returns

A clone of the transformation

See also

setTransformation()

Definition at line 265 of file qwt_scale_engine.cpp.

14.136.5.19 upperMargin() `double QwtScaleEngine::upperMargin ( ) const`

Returns

the margin at the upper end of the scale The default margin is 0.

See also

setMargins()

Definition at line 291 of file qwt_scale_engine.cpp.

14.137 QwtScaleMap Class Reference

A scale map.

`#include <qwt_scale_map.h>`

Public Member Functions

- QwtScaleMap ()
  Constructor.
- QwtScaleMap (const QwtScaleMap &)
  Copy constructor.
- ~QwtScaleMap ()
- QwtScaleMap & operator= (const QwtScaleMap &)
  Assignment operator.
- void setTransformation (QwtTransform *)
- const QwtTransform * transformation () const
  Get the transformation.
- void setPaintInterval (double p1, double p2)
  Specify the borders of the paint device interval.
- void setScaleInterval (double s1, double s2)
  Specify the borders of the scale interval.
- double transform (double s) const
- double invTransform (double p) const
- double p1 () const
- double p2 () const
- double s1 () const
- double s2 () const
- double pDist () const
- double sDist () const
- bool isInverting () const

Static Public Member Functions

- static QRectF transform (const QwtScaleMap &, const QwtScaleMap &, const QRectF &)
- static QRectF invTransform (const QwtScaleMap &, const QwtScaleMap &, const QRectF &)
- static QPointF transform (const QwtScaleMap &, const QwtScaleMap &, const QPointF &)
- static QPointF invTransform (const QwtScaleMap &, const QwtScaleMap &, const QPointF &)

14.137.1 Detailed Description

A scale map.

QwtScaleMap offers transformations from the coordinate system of a scale into the linear coordinate system of a paint device and vice versa.

Definition at line 26 of file qwt_scale_map.h.

14.137.2 Constructor & Destructor Documentation

14.137.2.1 QwtScaleMap() `QwtScaleMap::QwtScaleMap ( )`

Constructor.

The scale and paint device intervals are both set to [0,1].

Definition at line 21 of file qwt_scale_map.cpp.

14.137.2.2 ~QwtScaleMap() `QwtScaleMap::~QwtScaleMap ( )`

Destructor

Definition at line 49 of file qwt_scale_map.cpp.

14.137.3 Member Function Documentation

14.137.3.1 invTransform() [1/3]

```
QPointF QwtScaleMap::invTransform (
    const QwtScaleMap & xMap,
    const QwtScaleMap & yMap,
    const QPointF & pos ) [static]
```

Transform a rectangle from paint to scale coordinates

Parameters

xMap	X map
yMap	Y map
pos	Position in paint coordinates

Returns

Position in scale coordinates

See also

transform()

Definition at line 187 of file qwt_scale_map.cpp.

14.137.3.2 invTransform() [2/3] QRectF QwtScaleMap::invTransform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRectF & *rect*) [static]

Transform a rectangle from paint to scale coordinates

Parameters

xMap	X map
yMap	Y map
rect	Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

transform()

Definition at line 224 of file qwt_scale_map.cpp.

14.137.3.3 invTransform() [3/3] double QwtScaleMap::invTransform (double *p*) const [inline]

Transform an paint device value into a value in the interval of the scale.

Parameters

p	Value relative to the coordinates of the paint device

Returns

Transformed value

See also

transform()

Definition at line 154 of file qwt_scale_map.h.

14.137.3.4 isInverting() bool QwtScaleMap::isInverting ( ) const [inline]

Returns

True, when (p1() < p2()) != (s1() < s2())

Definition at line 164 of file qwt_scale_map.h.

14.137.3.5 p1() `double QwtScaleMap::p1 ( ) const [inline]`

Returns

First border of the paint interval

Definition at line 99 of file qwt_scale_map.h.

14.137.3.6 p2() `double QwtScaleMap::p2 ( ) const [inline]`

Returns

Second border of the paint interval

Definition at line 107 of file qwt_scale_map.h.

14.137.3.7 pDist() `double QwtScaleMap::pDist ( ) const [inline]`

Returns

qwtAbs(p2() - p1())

Definition at line 115 of file qwt_scale_map.h.

14.137.3.8 s1() `double QwtScaleMap::s1 ( ) const [inline]`

Returns

First border of the scale interval

Definition at line 83 of file qwt_scale_map.h.

14.137.3.9 s2() `double QwtScaleMap::s2 ( ) const [inline]`

Returns

Second border of the scale interval

Definition at line 91 of file qwt_scale_map.h.

14.137.3.10 sDist() `double QwtScaleMap::sDist ( ) const [inline]`

Returns

qwtAbs(s2() - s1())

Definition at line 123 of file qwt_scale_map.h.

14.137.3.11 setPaintInterval() `void QwtScaleMap::setPaintInterval (`

```
double p1,
double p2 )
```

Specify the borders of the paint device interval.

Parameters

p1	first border
p2	second border

Definition at line 119 of file qwt_scale_map.cpp.

14.137.3.12 setScaleInterval() `void QwtScaleMap::setScaleInterval (`

```
double s1,
double s2 )
```

Specify the borders of the scale interval.

Parameters

s1	first border
s2	second border

Warning

scales might be aligned to transformation depending boundaries

Definition at line 100 of file qwt_scale_map.cpp.

14.137.3.13 setTransformation() `void QwtScaleMap::setTransformation (`

```
QwtTransform * transform )
```

Initialize the map with a transformation

Definition at line 76 of file qwt_scale_map.cpp.

14.137.3.14 transform() [1/3] `QPointF QwtScaleMap::transform (`

```
const QwtScaleMap & xMap,
const QwtScaleMap & yMap,
const QPointF & pos )  [static]
```

Transform a point from scale to paint coordinates

Parameters

xMap	X map
yMap	Y map
pos	Position in scale coordinates

Returns

Position in paint coordinates

See also

invTransform()

Definition at line 206 of file qwt_scale_map.cpp.

14.137.3.15 transform() [2/3] QRectF QwtScaleMap::transform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRectF & *rect*) [static]

Transform a rectangle from scale to paint coordinates

Parameters

xMap	X map
yMap	Y map
rect	Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

invTransform()

Definition at line 153 of file qwt_scale_map.cpp.

14.137.3.16 transform() [3/3] double 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

Parameters

s	Value relative to the coordinates of the scale

Returns

Transformed value

See also

invTransform()

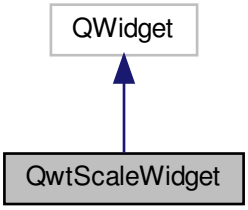
Definition at line 137 of file qwt_scale_map.h.

14.138 QwtScaleWidget Class Reference

A Widget which contains a scale.

`#include <qwt_scale_widget.h>`

Inheritance diagram for QwtScaleWidget:



Public Types

- enum LayoutFlag { TitleInverted = 1 }
  Layout flags of the title.
- typedef QFlags< LayoutFlag > LayoutFlags

Signals

- void scaleDivChanged ()
  Signal emitted, whenever the scale division changes.

Public Member Functions

- QwtScaleWidget (QWidget *parent=NULL)
  Create a scale with the position QwtScaleWidget::Left.
- QwtScaleWidget (QwtScaleDraw::Alignment, QWidget *parent=NULL)
  Constructor.
- virtual ~QwtScaleWidget ()
  Destructor.
- void setTitle (const QString &title)
- void setTitle (const QwtText &title)
- QwtText title () const

- void setLayoutFlag (LayoutFlag, bool on)
- bool testLayoutFlag (LayoutFlag) const
- void setBorderDist (int dist1, int dist2)
- int startBorderDist () const
- int endBorderDist () const
- void getBorderDistHint (int &start, int &end) const

  Calculate a hint for the border distances.
- void getMinBorderDist (int &start, int &end) const
- void setMinBorderDist (int start, int end)
- void setMargin (int)

  Specify the margin to the colorBar/base line.
- int margin () const
- void setSpacing (int)

  Specify the distance between color bar, scale and title.
- int spacing () const
- void setScaleDiv (const QwtScaleDiv &)

  Assign a scale division.
- void setTransformation (QwtTransform *)
- void setScaleDraw (QwtScaleDraw *)
- const QwtScaleDraw * scaleDraw () const
- QwtScaleDraw * scaleDraw ()
- void setLabelAlignment (Qt::Alignment)

  Change the alignment for the labels.
- void setLabelRotation (double rotation)

  Change the rotation for the labels. See QwtScaleDraw::setLabelRotation().
- void setColorBarEnabled (bool)
- bool isColorBarEnabled () const
- void setColorBarWidth (int)
- int colorBarWidth () const
- void setColorMap (const QwtInterval &, QwtColorMap *)
- QwtInterval colorBarInterval () const
- const QwtColorMap * colorMap () const
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override
- int titleHeightForWidth (int width) const

  Find the height of the title for a given width.
- int 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.
- void drawColorBar (QPainter *, const QRectF &) const
- void drawTitle (QPainter *, QwtScaleDraw::Alignment, const QRectF &rect) const
- void setAlignment (QwtScaleDraw::Alignment)
- QwtScaleDraw::Alignment alignment () const
- QRectF colorBarRect (const QRectF &) const

Protected Member Functions

- virtual void paintEvent (QPaintEvent *) override

  paintEvent
- virtual void resizeEvent (QResizeEvent *) override
- virtual void changeEvent (QEvent *) override
- void draw (QPainter *) const

  draw the scale
- void scaleChange ()

  Notify a change of the scale.
- void layoutScale (bool update_geometry=true)

14.138.1 Detailed Description

A Widget which contains a scale.

This Widget can be used to decorate composite widgets with a scale.

Definition at line 34 of file qwt_scale_widget.h.

14.138.2 Member Typedef Documentation

14.138.2.1 LayoutFlags `typedef QFlags<LayoutFlag > QwtScaleWidget::LayoutFlags`

An ORed combination of LayoutFlag values.

Definition at line 49 of file qwt_scale_widget.h.

14.138.3 Member Enumeration Documentation

14.138.3.1 LayoutFlag `enum QwtScaleWidget::LayoutFlag`

Layout flags of the title.

Enumerator

TitleInverted	The title of vertical scales is painted from top to bottom. Otherwise it is painted from bottom to top.

Definition at line 40 of file qwt_scale_widget.h.

14.138.4 Constructor & Destructor Documentation

14.138.4.1 QwtScaleWidget() [1/2] `QwtScaleWidget::QwtScaleWidget ( QWidget * parent = NULL ) [explicit]`

Create a scale with the position QwtScaleWidget::Left.

Parameters

parent	Parent widget

Definition at line 68 of file qwt_scale_widget.cpp.

14.138.4.2 QwtScaleWidget() [2/2] QwtScaleWidget::QwtScaleWidget (QwtScaleDraw::Alignment *align*, QWidget * *parent* = *NULL*) [explicit]

Constructor.

Parameters

align	Alignment.
parent	Parent widget

Definition at line 79 of file qwt_scale_widget.cpp.

14.138.5 Member Function Documentation

14.138.5.1 alignment() QwtScaleDraw::Alignment QwtScaleWidget::alignment ( ) const

Returns

position

See also

setPosition()

Definition at line 234 of file qwt_scale_widget.cpp.

14.138.5.2 changeEvent() void QwtScaleWidget::changeEvent (QEvent * *event*) [override], [protected], [virtual]

Change Event handler

Parameters

event	Change event

Invalidates internal caches if necessary

Definition at line 519 of file qwt_scale_widget.cpp.

14.138.5.3 colorBarInterval() `QwtInterval QwtScaleWidget::colorBarInterval ( ) const`

Returns

Value interval for the color bar

See also

setColorMap(), colorMap()

Definition at line 941 of file qwt_scale_widget.cpp.

14.138.5.4 colorBarRect() `QRectF QwtScaleWidget::colorBarRect ( const QRectF & rect ) const`

Calculate the the rectangle for the color bar

Parameters

rect	Bounding rectangle for all components of the scale

Returns

Rectangle for the color bar

Definition at line 462 of file qwt_scale_widget.cpp.

14.138.5.5 colorBarWidth() `int QwtScaleWidget::colorBarWidth ( ) const`

Returns

Width of the color bar

See also

setColorBarEnabled(), setColorBarEnabled()

Definition at line 932 of file qwt_scale_widget.cpp.

14.138.5.6 colorMap() `const QwtColorMap * QwtScaleWidget::colorMap ( ) const`

Returns

Color map

See also

setColorMap(), colorBarInterval()

Definition at line 974 of file qwt_scale_widget.cpp.

14.138.5.7 dimForLength()

```
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

length	width for horizontal, height for vertical scales
scaleFont	Font of the scale

Returns

height for horizontal, width for vertical scales

Definition at line 781 of file qwt_scale_widget.cpp.

14.138.5.8 drawColorBar()

```
void QwtScaleWidget::drawColorBar (
    QPainter * painter,
    const QRectF & rect ) const
```

Draw the color bar of the scale widget

Parameters

painter	Painter
rect	Bounding rectangle for the color bar

See also

setColorBarEnabled()

Definition at line 624 of file qwt_scale_widget.cpp.

14.138.5.9 drawTitle()

```
void QwtScaleWidget::drawTitle (
    QPainter * painter,
    QwtScaleDraw::Alignment align,
    const QRectF & rect ) const
```

Rotate and paint a title according to its position into a given rectangle.

Parameters

painter	Painter
align	Alignment
rect	Bounding rectangle

Definition at line 644 of file qwt_scale_widget.cpp.

14.138.5.10 endBorderDist()

```
int QwtScaleWidget::endBorderDist ( ) const
```

Returns

end border distance

See also

setBorderDist()

Definition at line 389 of file qwt_scale_widget.cpp.

14.138.5.11 getBorderDistHint()

```
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 an the minimum border distance is returned.

Parameters

start	Return parameter for the border width at the beginning of the scale
end	Return parameter for the border width at the end of the scale

Warning

- The minimum border distance depends on the font.

See also

setMinBorderDist(), getMinBorderDist(), setBorderDist()

Definition at line 814 of file qwt_scale_widget.cpp.

14.138.5.12 getMinBorderDist()

```
void QwtScaleWidget::getMinBorderDist (
        int & start,
        int & end ) const
```

Get the minimum value for the distances of the scale's endpoints from the widget borders.

Parameters

start	Return parameter for the border width at the beginning of the scale
end	Return parameter for the border width at the end of the scale

See also

setMinBorderDist(), getBorderDistHint()

Definition at line 852 of file qwt_scale_widget.cpp.

14.138.5.13 isColorBarEnabled()

```
bool QwtScaleWidget::isColorBarEnabled ( ) const
```

Returns

true, when the color bar is enabled

See also

setColorBarEnabled(), setColorBarWidth()

Definition at line 907 of file qwt_scale_widget.cpp.

14.138.5.14 layoutScale()

```
void QwtScaleWidget::layoutScale (
        bool update_geometry = true )  [protected]
```

Recalculate the scale's geometry and layout based on the current geometry and fonts.

Parameters

update_geometry	Notify the layout system and call update to redraw the scale

Definition at line 547 of file qwt_scale_widget.cpp.

14.138.5.15 margin() `int QwtScaleWidget::margin ( ) const`

Returns

margin

See also

setMargin()

Definition at line 398 of file qwt_scale_widget.cpp.

14.138.5.16 minimumSizeHint() `QSize QwtScaleWidget::minimumSizeHint ( ) const [override], [virtual]`

Returns

a minimum size hint

Definition at line 732 of file qwt_scale_widget.cpp.

14.138.5.17 resizeEvent() `void QwtScaleWidget::resizeEvent ( QResizeEvent * event ) [override], [protected], [virtual]`

Event handler for resize events

Parameters

event	Resize event

Definition at line 533 of file qwt_scale_widget.cpp.

14.138.5.18 scaleChange() `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.

Definition at line 716 of file qwt_scale_widget.cpp.

14.138.5.19 scaleDraw() [1/2] QwtScaleDraw * QwtScaleWidget::scaleDraw ( )

Returns

scaleDraw of this scale

See also

QwtScaleDraw::setScaleDraw()

Definition at line 362 of file qwt_scale_widget.cpp.

14.138.5.20 scaleDraw() [2/2] const QwtScaleDraw * QwtScaleWidget::scaleDraw ( ) const

Returns

scaleDraw of this scale

See also

setScaleDraw(), QwtScaleDraw::setScaleDraw()

Definition at line 353 of file qwt_scale_widget.cpp.

14.138.5.21 setAlignment() void QwtScaleWidget::setAlignment (QwtScaleDraw::Alignment *alignment*)

Change the alignment

Parameters

alignment	New alignment

See also

alignment()

Definition at line 209 of file qwt_scale_widget.cpp.

14.138.5.22 setBorderDist() 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

dist1	Left or top Distance
dist2	Right or bottom distance

See also

borderDist()

Definition at line 250 of file qwt_scale_widget.cpp.

14.138.5.23 setColorBarEnabled() `void QwtScaleWidget::setColorBarEnabled ( bool on )`

En/disable a color bar associated to the scale

See also

isColorBarEnabled(), setColorBarWidth()

Definition at line 894 of file qwt_scale_widget.cpp.

14.138.5.24 setColorBarWidth() `void QwtScaleWidget::setColorBarWidth ( int width )`

Set the width of the color bar

Parameters

width	Width

See also

colorBarWidth(), setColorBarEnabled()

Definition at line 918 of file qwt_scale_widget.cpp.

14.138.5.25 setColorMap() `void QwtScaleWidget::setColorMap ( const QwtInterval & interval, QwtColorMap * colorMap )`

Set the color map and value interval, that are used for displaying the color bar.

Parameters

interval	Value interval
colorMap	Color map

See also

colorMap(), colorBarInterval()

Definition at line 955 of file qwt_scale_widget.cpp.

14.138.5.26 setLabelAlignment() `void QwtScaleWidget::setLabelAlignment ( Qt::Alignment alignment )`

Change the alignment for the labels.

See also

QwtScaleDraw::setLabelAlignment(), setLabelRotation()

Definition at line 295 of file qwt_scale_widget.cpp.

14.138.5.27 setLabelRotation() `void QwtScaleWidget::setLabelRotation ( double rotation )`

Change the rotation for the labels. See QwtScaleDraw::setLabelRotation().

Parameters

rotation	Rotation

See also

QwtScaleDraw::setLabelRotation(), setLabelFlags()

Definition at line 308 of file qwt_scale_widget.cpp.

14.138.5.28 setLayoutFlag() `void QwtScaleWidget::setLayoutFlag ( LayoutFlag flag, bool on )`

Toggle an layout flag

Parameters

flag	Layout flag
on	true/false

See also

testLayoutFlag(), LayoutFlag

Definition at line 141 of file qwt_scale_widget.cpp.

14.138.5.29 setMargin() `void QwtScaleWidget::setMargin ( int margin )`

Specify the margin to the colorBar/base line.

Parameters

margin	Margin

See also

margin()

Definition at line 265 of file qwt_scale_widget.cpp.

14.138.5.30 setMinBorderDist() `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

start	Minimum for the start border
end	Minimum for the end border

See also

getMinBorderDist(), getBorderDistHint()

Definition at line 835 of file qwt_scale_widget.cpp.

14.138.5.31 setScaleDiv() `void QwtScaleWidget::setScaleDiv ( const QwtScaleDiv & scaleDiv )`

Assign a scale division.

The scale division determines where to set the tick marks.

Parameters

scaleDiv	Scale Division

See also

For more information about scale divisions, see QwtScaleDiv.

Definition at line 866 of file qwt_scale_widget.cpp.

14.138.5.32 setScaleDraw() `void QwtScaleWidget::setScaleDraw ( QwtScaleDraw * scaleDraw )`

Set a scale draw

scaleDraw has to be created with new and will be deleted in ~QwtScaleWidget() or the next call of setScaleDraw(). scaleDraw will be initialized with the attributes of the previous scaleDraw object.

Parameters

scaleDraw	ScaleDraw object

See also

scaleDraw()

Definition at line 325 of file qwt_scale_widget.cpp.

14.138.5.33 setSpacing() `void QwtScaleWidget::setSpacing ( int spacing )`

Specify the distance between color bar, scale and title.

Parameters

spacing	Spacing

See also

spacing()

Definition at line 280 of file qwt_scale_widget.cpp.

14.138.5.34 setTitle() [1/2] void QwtScaleWidget::setTitle (const QString & *title*)

Give title new text contents

Parameters

title	New title

See also

title(), setTitle(const QwtText &);

Definition at line 172 of file qwt_scale_widget.cpp.

14.138.5.35 setTitle() [2/2] void QwtScaleWidget::setTitle (const QwtText & *title*)

Give title new text contents

Parameters

title	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.

Definition at line 190 of file qwt_scale_widget.cpp.

14.138.5.36 setTransformation() void QwtScaleWidget::setTransformation (QwtTransform * *transformation*)

Set the transformation

Parameters

transformation	Transformation

See also

QwtAbstractScaleDraw::scaleDraw(), QwtScaleMap

Definition at line 884 of file qwt_scale_widget.cpp.

14.138.5.37 sizeHint() `QSize QwtScaleWidget::sizeHint ( ) const [override], [virtual]`

Returns

a size hint

Definition at line 724 of file qwt_scale_widget.cpp.

14.138.5.38 spacing() `int QwtScaleWidget::spacing ( ) const`

Returns

distance between scale and title

See also

setMargin()

Definition at line 407 of file qwt_scale_widget.cpp.

14.138.5.39 startBorderDist() `int QwtScaleWidget::startBorderDist ( ) const`

Returns

start border distance

See also

setBorderDist()

Definition at line 380 of file qwt_scale_widget.cpp.

14.138.5.40 testLayoutFlag() `bool QwtScaleWidget::testLayoutFlag ( LayoutFlag flag ) const`

Test a layout flag

Parameters

flag	Layout flag

Returns

true/false

See also

setLayoutFlag(), LayoutFlag

Definition at line 161 of file qwt_scale_widget.cpp.

14.138.5.41 title() `QwtText QwtScaleWidget::title ( ) const`

Returns

title

See also

setTitle()

Definition at line 371 of file qwt_scale_widget.cpp.

14.138.5.42 titleHeightForWidth() `int QwtScaleWidget::titleHeightForWidth ( int width ) const`

Find the height of the title for a given width.

Parameters

width	Width

Returns

height Height

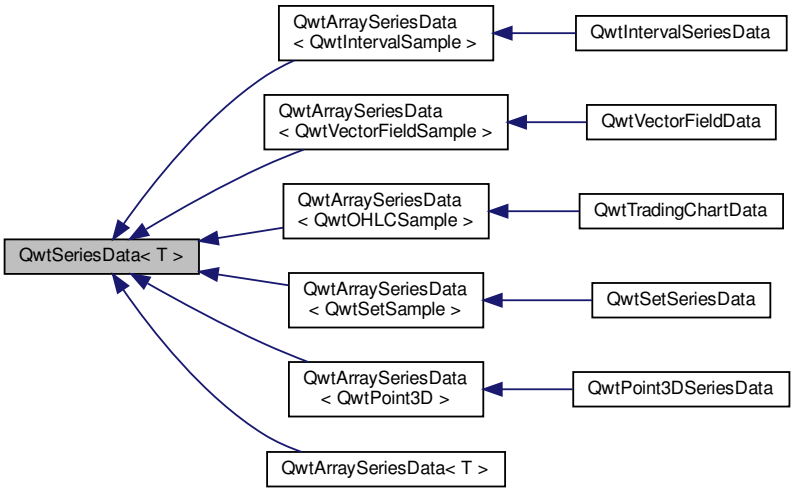
Definition at line 767 of file qwt_scale_widget.cpp.

14.139 QwtSeriesData< T > Class Template Reference

Abstract interface for iterating over samples.

#include <qwt_series_data.h>

Inheritance diagram for QwtSeriesData< T >:



Public Member Functions

- QwtSeriesData ()
  Constructor.
- virtual ~QwtSeriesData ()
  Destructor.
- virtual size_t size () const =0
- virtual T sample (size_t i) const =0
- virtual QRectF boundingRect () const =0
- virtual void setRectOfInterest (const QRectF &rect)

Protected Attributes

- QRectF cachedBoundingRect
  Can be used to cache a calculated bounding rectangle.

14.139.1 Detailed Description

template<typename T>
class QwtSeriesData< T >

Abstract interface for iterating over samples.

Qwt offers several implementations of the QwtSeriesData API, but in situations, where data of an application specific format needs to be displayed, without having to copy it, it is recommended to implement an individual data access.

A subclass of QwtSeriesData<QPointF> must implement:

- size()
  Should return number of data points.
- sample()
  Should return values x and y values of the sample at specific position as QPointF object.
- boundingRect()
  Should return the bounding rectangle of the data series. It is used for autoscaling and might help certain algorithms for displaying the data. You can use qwtBoundingRect() for an implementation but often it is possible to implement a more efficient algorithm depending on the characteristics of the series. The member cachedBoundingRect is intended for caching the calculated rectangle.

Definition at line 49 of file qwt_series_data.h.

14.139.2 Member Function Documentation

14.139.2.1 boundingRect() template<typename T >

```
virtual QRectF QwtSeriesData< T >::boundingRect ( ) const  [pure virtual]
```

Calculate the bounding rect of all samples

The bounding rect is necessary for autoscaling and can be used for a couple of painting optimizations.

qwtBoundingRect(...) offers slow implementations iterating over the samples. For large sets it is recommended to implement something faster f.e. by caching the bounding rectangle.

Returns

Bounding rectangle

Implemented in QwtTradingChartData, QwtVectorFieldData, QwtSetSeriesData, QwtIntervalSeriesData, QwtPoint3DSeriesData, QwtPointSeriesData, and QwtSyntheticPointData.

14.139.2.2 sample() template<typename T >

```
virtual T QwtSeriesData< T >::sample (
            size_t i ) const  [pure virtual]
```

Return a sample

Parameters

i	Index

Returns

Sample at position i

Implemented in QwtArraySeriesData< T >, QwtArraySeriesData< QwtIntervalSample >, QwtArraySeriesData< QwtVectorFieldSam
QwtArraySeriesData< QwtOHLCSample >, QwtArraySeriesData< QPointF >, QwtArraySeriesData< QwtSetSample >,
QwtArraySeriesData< QwtPoint3D >, QwtSyntheticPointData, QwtCPointerValueData< T >, QwtValuePointData< T >,
QwtCPointerData< T >, and QwtPointArrayData< T >.

14.139.2.3 setRectOfInterest() template<typename T >

```
void QwtSeriesData< T >::setRectOfInterest (
            const QRectF & rect )  [virtual]
```

Set a the "rect of interest"

QwtPlotSeriesItem defines the current area of the plot canvas as "rectangle of interest" (QwtPlotSeriesItem::updateScaleDiv()). It can be used to implement different levels of details.

The default implementation does nothing.

Parameters

rect	Rectangle of interest

Reimplemented in QwtSyntheticPointData.

Definition at line 124 of file qwt_series_data.h.

14.139.2.4 size() template<typename T >

```
virtual size_t QwtSeriesData< T >::size ( ) const  [pure virtual]
```

Returns

Number of samples

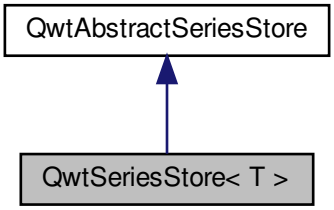
Implemented in QwtArraySeriesData< T >, QwtArraySeriesData< QwtIntervalSample >, QwtArraySeriesData< QwtVectorFieldSam
QwtArraySeriesData< QwtOHLCSample >, QwtArraySeriesData< QPointF >, QwtArraySeriesData< QwtSetSample >,
QwtArraySeriesData< QwtPoint3D >, QwtSyntheticPointData, QwtCPointerValueData< T >, QwtValuePointData< T >,
QwtCPointerData< T >, and QwtPointArrayData< T >.

14.140 QwtSeriesStore< T > Class Template Reference

Class storing a QwtSeriesData object.

```
#include <qwt_series_store.h>
```

Inheritance diagram for QwtSeriesStore< T >:



Public Member Functions

- QwtSeriesStore ()

  Constructor The store contains no series.
- ~QwtSeriesStore ()

  Destructor.
- void setData (QwtSeriesData< T > *series)
- QwtSeriesData< T > * data ()
- const QwtSeriesData< T > * data () const
- T sample (int index) const
- virtual size_t dataSize () const override
- virtual QRectF dataRect () const override
- virtual void setRectOfInterest (const QRectF &rect) override
- QwtSeriesData< T > * swapData (QwtSeriesData< T > *series)

Additional Inherited Members

14.140.1 Detailed Description

template<typename T>
class QwtSeriesStore< T >

Class storing a QwtSeriesData object.

QwtSeriesStore and QwtPlotSeriesItem are intended as base classes for all plot items iterating over a series of samples. Both classes share a virtual base class (QwtAbstractSeriesStore) to bridge between them.

QwtSeriesStore offers the template based part for the plot item API, so that QwtPlotSeriesItem can be derived without any hassle with templates.

Definition at line 66 of file qwt_series_store.h.

14.140.2 Member Function Documentation

14.140.2.1 data() [1/2] template<typename T >

QwtSeriesData< T > * QwtSeriesStore< T >::data [inline]

Returns

the the series data

Definition at line 146 of file qwt_series_store.h.

14.140.2.2 data() [2/2] template<typename T >

const QwtSeriesData< T > * QwtSeriesStore< T >::data [inline]

Returns

the the series data

Definition at line 152 of file qwt_series_store.h.

14.140.2.3 dataRect() template<typename T >

QRectF QwtSeriesStore< T >::dataRect [override], [virtual]

Returns

Bounding rectangle of the series or an invalid rectangle, when no series is stored

See also

QwtSeriesData<T>::boundingRect()

Implements QwtAbstractSeriesStore.

Definition at line 184 of file qwt_series_store.h.

14.140.2.4 dataSize() template<typename T >

size_t QwtSeriesStore< T >::dataSize [override], [virtual]

Returns

Number of samples of the series

See also

setData(), QwtSeriesData<T>::size()

Implements QwtAbstractSeriesStore.

Definition at line 175 of file qwt_series_store.h.

14.140.2.5 sample() template<typename T >

T QwtSeriesStore< T >::sample (
int *index*) const [inline]

Parameters

index	Index

Returns

Sample at position index

Definition at line 158 of file qwt_series_store.h.

14.140.2.6 setData()

```
template<typename T >
void QwtSeriesStore< T >::setData (
            QwtSeriesData< T > * series )
```

Assign a series of samples

Parameters

series	Data

Warning

The item takes ownership of the data object, deleting it when its not used anymore.

Definition at line 164 of file qwt_series_store.h.

14.140.2.7 setRectOfInterest()

```
template<typename T >
void QwtSeriesStore< T >::setRectOfInterest (
            const QRectF & rect )  [override], [virtual]
```

Set a the "rect of interest" for the series

Parameters

rect	Rectangle of interest

See also

QwtSeriesData<T>::setRectOfInterest()

Implements QwtAbstractSeriesStore.

Definition at line 193 of file qwt_series_store.h.

14.140.2.8 swapData() template<typename T >

```
QwtSeriesData< T > * QwtSeriesStore< T >::swapData (
            QwtSeriesData< T > * series )
```

Replace a series without deleting the previous one

Parameters

series	New series

Returns

Previously assigned series

Definition at line 200 of file qwt_series_store.h.

14.141 QwtSetSample Class Reference

A sample of the types (x1...xn, y) or (x, y1..yn)

```
#include <qwt_samples.h>
```

Public Member Functions

- QwtSetSample ()
- QwtSetSample (double, const QVector< double > &=QVector< double >())
- bool operator== (const QwtSetSample &other) const

  Compare operator.
- bool operator!= (const QwtSetSample &other) const

  Compare operator.
- double added () const

Public Attributes

- double value

  value
- QVector< double > set

  Vector of values associated to value.

14.141.1 Detailed Description

A sample of the types (x1...xn, y) or (x, y1..yn)

Definition at line 73 of file qwt_samples.h.

14.141.2 Constructor & Destructor Documentation

14.141.2.1 QwtSetSample() [1/2] QwtSetSample::QwtSetSample ( ) [inline]

Constructor The value is set to 0.0

Definition at line 95 of file qwt_samples.h.

14.141.2.2 QwtSetSample() [2/2] QwtSetSample::QwtSetSample (double v, const QVector< double > & s = QVector< double >( )) [inline], [explicit]

Constructor

Parameters

v	Value
s	Set of values

Definition at line 106 of file qwt_samples.h.

14.141.3 Member Function Documentation

14.141.3.1 added() double QwtSetSample::added ( ) const [inline]

Returns

All values of the set added

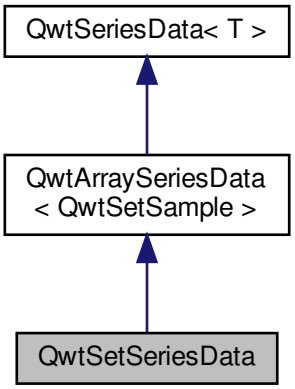
Definition at line 125 of file qwt_samples.h.

14.142 QwtSetSeriesData Class Reference

Interface for iterating over an array of samples.

`#include <qwt_series_data.h>`

Inheritance diagram for QwtSetSeriesData:



Public Member Functions

- QwtSetSeriesData (const QVector< QwtSetSample > &=QVector< QwtSetSample >())
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.

Additional Inherited Members

14.142.1 Detailed Description

Interface for iterating over an array of samples.

Definition at line 239 of file qwt_series_data.h.

14.142.2 Constructor & Destructor Documentation

14.142.2.1 QwtSetSeriesData() `QwtSetSeriesData::QwtSetSeriesData ( const QVector< QwtSetSample > & samples = QVector< QwtSetSample >( ) )`

Constructor

Parameters

samples	Samples

Definition at line 353 of file qwt_series_data.cpp.

14.142.3 Member Function Documentation

14.142.3.1 boundingRect() `QRectF QwtSetSeriesData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

Implements QwtSeriesData< T >.

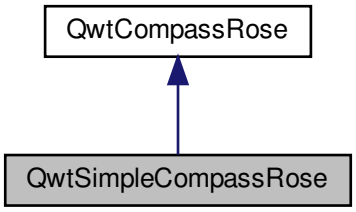
Definition at line 366 of file qwt_series_data.cpp.

14.143 QwtSimpleCompassRose Class Reference

A simple rose for QwtCompass.

`#include <qwt_compass_rose.h>`

Inheritance diagram for QwtSimpleCompassRose:



Public Member Functions

- QwtSimpleCompassRose (int numThorns=8, int numThornLevels=-1)
- virtual ~QwtSimpleCompassRose ()

  Destructor.
- void setWidth (double)
- double width () const
- void setNumThorns (int)
- int numThorns () const
- void setNumThornLevels (int)
- int numThornLevels () const
- void setShrinkFactor (double factor)
- double shrinkFactor () const
- virtual void draw (QPainter *, const QPointF &center, double radius, double north, QPalette::Color↩ Group=QPalette::Active) const override

Static Public Member Functions

- static void drawRose (QPainter ∗, const QPalette &, const QPointF &center, double radius, double north, double width, int numThorns, int numThornLevels, double shrinkFactor)

14.143.1 Detailed Description

A simple rose for QwtCompass.

Definition at line 52 of file qwt_compass_rose.h.

14.143.2 Constructor & Destructor Documentation

14.143.2.1 QwtSimpleCompassRose() `QwtSimpleCompassRose::QwtSimpleCompassRose ( int numThorns = 8, int numThornLevels = -1 )`

Constructor

Parameters

numThorns	Number of thorns
numThornLevels	Number of thorn levels

Definition at line 78 of file qwt_compass_rose.cpp.

14.143.3 Member Function Documentation

14.143.3.1 draw() `void QwtSimpleCompassRose::draw ( QPainter * painter, const QPointF & center, double radius, double north, QPalette::ColorGroup cg = QPalette::Active ) const [override], [virtual]`

Draw the rose

Parameters

painter	Painter
center	Center point
radius	Radius of the rose
north	Position
cg	Color group

Implements QwtCompassRose.

Definition at line 131 of file qwt_compass_rose.cpp.

14.143.3.2 drawRose()

```
void QwtSimpleCompassRose::drawRose (
    QPainter * painter,
    const QPalette & palette,
    const QPointF & center,
    double radius,
    double north,
    double width,
    int numThorns,
    int numThornLevels,
    double shrinkFactor ) [static]
```

Draw the rose

Parameters

painter	Painter
palette	Palette
center	Center of the rose
radius	Radius of the rose
north	Position pointing to north
width	Width of the rose
numThorns	Number of thorns
numThornLevels	Number of thorn levels
shrinkFactor	Factor to shrink the thorns with each level

Definition at line 154 of file qwt_compass_rose.cpp.

14.143.3.3 numThornLevels()

```
int QwtSimpleCompassRose::numThornLevels ( ) const
```

Returns

Number of thorn levels

See also

setNumThorns(), setNumThornLevels()

Definition at line 293 of file qwt_compass_rose.cpp.

14.143.3.4 numThorns() `int QwtSimpleCompassRose::numThorns ( ) const`

Returns

Number of thorns

See also

setNumThorns(), setNumThornLevels()

Definition at line 273 of file qwt_compass_rose.cpp.

14.143.3.5 setNumThornLevels() `void QwtSimpleCompassRose::setNumThornLevels ( int numThornLevels )`

Set the of thorns levels

Parameters

numThornLevels	Number of thorns levels

See also

setNumThorns(), numThornLevels()

Definition at line 284 of file qwt_compass_rose.cpp.

14.143.3.6 setNumThorns() `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

numThorns	Number of thorns

See also

numThorns(), setNumThornLevels()

Definition at line 258 of file qwt_compass_rose.cpp.

14.143.3.7 setShrinkFactor() `void QwtSimpleCompassRose::setShrinkFactor ( double factor )`

Set the Factor how to shrink the thorns with each level The default value is 0.9.

Parameters

factor	Shrink factor

See also

shrinkFactor()

Definition at line 108 of file qwt_compass_rose.cpp.

14.143.3.8 setWidth() `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

width	Width

Definition at line 232 of file qwt_compass_rose.cpp.

14.143.3.9 shrinkFactor() `double QwtSimpleCompassRose::shrinkFactor ( ) const`

Returns

Factor how to shrink the thorns with each level

See also

setShrinkFactor()

Definition at line 117 of file qwt_compass_rose.cpp.

14.143.3.10 width() `double QwtSimpleCompassRose::width ( ) const`

Returns

Width of the rose

See also

setWidth()

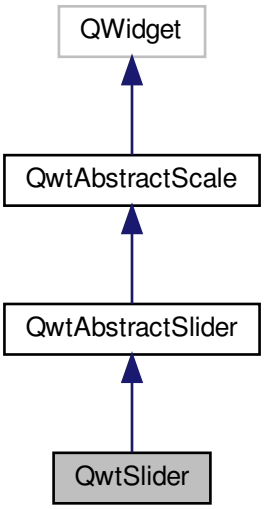
Definition at line 246 of file qwt_compass_rose.cpp.

14.144 QwtSlider Class Reference

The Slider Widget.

`#include <qwt_slider.h>`

Inheritance diagram for QwtSlider:



Public Types

- enum ScalePosition { NoScale , LeadingScale , TrailingScale }

Public Member Functions

- QwtSlider (QWidget ∗parent=NULL)
- QwtSlider (Qt::Orientation, QWidget ∗parent=NULL)
- virtual ∼QwtSlider ()

  Destructor.
- void setOrientation (Qt::Orientation)

  Set the orientation.
- Qt::Orientation orientation () const
- void setScalePosition (ScalePosition)

  Change the position of the scale.
- ScalePosition scalePosition () const
- void setTrough (bool)
- bool hasTrough () const
- void setGroove (bool)
- bool hasGroove () const
- void setHandleSize (const QSize &)

Set the slider's handle size.

- QSize handleSize () const
- void setBorderWidth (int)

  Change the slider's border width.
- int borderWidth () const
- void setSpacing (int)

  Change the spacing between trough and scale.
- int spacing () const
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override
- void setScaleDraw (QwtScaleDraw *)

  Set a scale draw.
- const QwtScaleDraw * scaleDraw () const
- void setUpdateInterval (int)

  Specify the update interval for automatic scrolling.
- int updateInterval () const

Protected Member Functions

- virtual double scrolledTo (const QPoint &) const override

  Determine the value for a new position of the slider handle.
- virtual bool isScrollPosition (const QPoint &) const override

  Determine what to do when the user presses a mouse button.
- virtual void drawSlider (QPainter *, const QRect &) const
- virtual void drawHandle (QPainter *, const QRect &, int pos) const
- virtual void mousePressEvent (QMouseEvent *) override
- virtual void mouseReleaseEvent (QMouseEvent *) override
- virtual void resizeEvent (QResizeEvent *) override
- virtual void paintEvent (QPaintEvent *) override
- virtual void changeEvent (QEvent *) override
- virtual void timerEvent (QTimerEvent *) override
- virtual bool event (QEvent *) override
- virtual void scaleChange () override

  Notify changed scale.
- QRect sliderRect () const
- QRect handleRect () const

Additional Inherited Members

14.144.1 Detailed Description

The Slider Widget.

QwtSlider is a slider widget which operates on an interval of type double. Its position is related to a scale showing the current value.

The slider can be customized by having a through, a groove - or both.

Definition at line 30 of file qwt_slider.h.

14.144.2 Member Enumeration Documentation

14.144.2.1 ScalePosition `enum QwtSlider::ScalePosition`

Position of the scale

See also

QwtSlider(), setScalePosition(), setOrientation()

Enumerator

NoScale	The slider has no scale.
LeadingScale	The scale is right of a vertical or below a horizontal slider.
TrailingScale	The scale is left of a vertical or above a horizontal slider.

Definition at line 54 of file qwt_slider.h.

14.144.3 Constructor & Destructor Documentation

14.144.3.1 QwtSlider() [1/2] `QwtSlider::QwtSlider ( QWidget * parent = NULL ) [explicit]`

Construct vertical slider in QwtSlider::Trough style with a scale to the left.

The scale is initialized to [0.0, 100.0] and the value set to 0.0.

Parameters

parent	Parent widget

See also

setOrientation(), setScalePosition(), setBackgroundStyle()

Definition at line 119 of file qwt_slider.cpp.

14.144.3.2 QwtSlider() [2/2] `QwtSlider::QwtSlider ( Qt::Orientation orientation, QWidget * parent = NULL ) [explicit]`

Construct a slider in QwtSlider::Trough style

When orientation is Qt::Vertical the scale will be aligned to the left - otherwise at the the top of the slider.

The scale is initialized to [0.0, 100.0] and the value set to 0.0.

Parameters

parent	Parent widget
orientation	Orientation of the slider.

Definition at line 136 of file qwt_slider.cpp.

14.144.4 Member Function Documentation

14.144.4.1 borderWidth() `int QwtSlider::borderWidth ( ) const`

Returns

the border width.

See also

setBorderWidth()

Definition at line 262 of file qwt_slider.cpp.

14.144.4.2 changeEvent() `void QwtSlider::changeEvent ( QEvent * event ) [override], [protected], [virtual]`

Handles QEvent::StyleChange and QEvent::FontChange events

Parameters

event	Change event

Reimplemented from QwtAbstractScale.

Definition at line 724 of file qwt_slider.cpp.

14.144.4.3 drawHandle() `void QwtSlider::drawHandle ( QPainter * painter, const QRect & handleRect, int pos ) const [protected], [virtual]`

Draw the thumb at a position

Parameters

painter	Painter
handleRect	Bounding rectangle of the handle
pos	Position of the handle marker in widget coordinates

Definition at line 473 of file qwt_slider.cpp.

14.144.4.4 drawSlider()

```
void QwtSlider::drawSlider (
    QPainter * painter,
    const QRect & sliderRect ) const  [protected], [virtual]
```

Draw the slider into the specified rectangle.

Parameters

painter	Painter
sliderRect	Bounding rectangle of the slider

Definition at line 415 of file qwt_slider.cpp.

14.144.4.5 event()

```
bool QwtSlider::event (
    QEvent * event )  [override], [protected], [virtual]
```

Qt event handler

Parameters

event	Event

Returns

true, if event was recognized and processed

Definition at line 712 of file qwt_slider.cpp.

14.144.4.6 handleRect()

```
QRect QwtSlider::handleRect ( ) const  [protected]
```

Returns

Bounding rectangle of the slider handle

Definition at line 990 of file qwt_slider.cpp.

14.144.4.7 handleSize() `QSize QwtSlider::handleSize ( ) const`

Returns

Size of the handle.

See also

setHandleSize()

Definition at line 326 of file qwt_slider.cpp.

14.144.4.8 hasGroove() `bool QwtSlider::hasGroove ( ) const`

Returns

True, when the groove is visible

See also

setGroove(), hasTrough()

Definition at line 904 of file qwt_slider.cpp.

14.144.4.9 hasTrough() `bool QwtSlider::hasTrough ( ) const`

Returns

True, when the trough is visible

See also

setTrough(), hasGroove()

Definition at line 875 of file qwt_slider.cpp.

14.144.4.10 isScrollPosition() `bool QwtSlider::isScrollPosition ( const QPoint & pos ) const [override], [protected], [virtual]`

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

Parameters

pos	Mouse position

Return values

True,when	handleRect() contains pos

See also

scrolledTo()

Implements QwtAbstractSlider.

Definition at line 503 of file qwt_slider.cpp.

14.144.4.11 minimumSizeHint() `QSize QwtSlider::minimumSizeHint ( ) const [override], [virtual]`

Returns

Minimum size hint

See also

sizeHint()

Definition at line 922 of file qwt_slider.cpp.

14.144.4.12 mousePressEvent() `void QwtSlider::mousePressEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse press event handler

Parameters

event	Mouse event

Reimplemented from QwtAbstractSlider.

Definition at line 547 of file qwt_slider.cpp.

14.144.4.13 mouseReleaseEvent()

```
void QwtSlider::mouseReleaseEvent (
        QMouseEvent * event )  [override], [protected], [virtual]
```

Mouse release event handler

Parameters

event	Mouse event

Reimplemented from QwtAbstractSlider.

Definition at line 606 of file qwt_slider.cpp.

14.144.4.14 orientation() `Qt::Orientation QwtSlider::orientation ( ) const`

Returns

Orientation

See also

setOrientation()

Definition at line 202 of file qwt_slider.cpp.

14.144.4.15 paintEvent() `void QwtSlider::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Qt paint event handler

Parameters

event	Paint event

Definition at line 675 of file qwt_slider.cpp.

14.144.4.16 resizeEvent() `void QwtSlider::resizeEvent ( QResizeEvent * event ) [override], [protected], [virtual]`

Qt resize event handler

Parameters

event	Resize event

Definition at line 700 of file qwt_slider.cpp.

14.144.4.17 scaleDraw() const QwtScaleDraw * QwtSlider::scaleDraw ( ) const

Returns

the scale draw of the slider

See also

setScaleDraw()

Definition at line 363 of file qwt_slider.cpp.

14.144.4.18 scalePosition() QwtSlider::ScalePosition QwtSlider::scalePosition ( ) const

Returns

Position of the scale

See also

setScalePosition()

Definition at line 230 of file qwt_slider.cpp.

14.144.4.19 scrolledTo() double QwtSlider::scrolledTo (const QPoint & *pos*) const [override], [protected], [virtual]

Determine the value for a new position of the slider handle.

Parameters

pos	Mouse position

Returns

Value for the mouse position

See also

isScrollPosition()

Implements QwtAbstractSlider.

Definition at line 526 of file qwt_slider.cpp.

14.144.4.20 setBorderWidth() `void QwtSlider::setBorderWidth ( int width )`

Change the slider's border width.

The border width is used for drawing the slider handle and the trough.

Parameters

width	Border width

See also

borderWidth()

Definition at line 244 of file qwt_slider.cpp.

14.144.4.21 setGroove() `void QwtSlider::setGroove ( bool on )`

En/Disable the groove

The slider can be customized by showing a groove for the handle.

Parameters

on	When true, the groove is visible

See also

hasGroove(), setThrough()

Definition at line 889 of file qwt_slider.cpp.

14.144.4.22 setHandleSize() `void QwtSlider::setHandleSize ( const QSize & size )`

Set the slider's handle size.

When the size is empty the slider handle will be painted with a default size depending on its orientation() and backgroundStyle().

Parameters

size	New size

See also

handleSize()

Definition at line 311 of file qwt_slider.cpp.

14.144.4.23 setOrientation() `void QwtSlider::setOrientation ( Qt::Orientation orientation )`

Set the orientation.

Parameters

orientation	Allowed values are Qt::Horizontal and Qt::Vertical.

See also

orientation(), scalePosition()

Definition at line 175 of file qwt_slider.cpp.

14.144.4.24 setScaleDraw() `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

scaleDraw	ScaleDraw object, that has to be created with new and will be deleted in ~QwtSlider() or the next call of setScaleDraw().

See also

scaleDraw()

Definition at line 344 of file qwt_slider.cpp.

14.144.4.25 setScalePosition() `void QwtSlider::setScalePosition ( ScalePosition scalePosition )`

Change the position of the scale.

Parameters

scalePosition	Position of the scale.

See also

ScalePosition, scalePosition()

Definition at line 213 of file qwt_slider.cpp.

14.144.4.26 setSpacing() `void QwtSlider::setSpacing ( int spacing )`

Change the spacing between trough and scale.

A spacing of 0 means, that the backbone of the scale is covered by the trough.

The default setting is 4 pixels.

Parameters

spacing	Number of pixels

See also

spacing();

Definition at line 278 of file qwt_slider.cpp.

14.144.4.27 setTrough() `void QwtSlider::setTrough ( bool on )`

En/Disable the trough

The slider can be customized by showing a trough for the handle.

Parameters

on	When true, the groove is visible

See also

hasTrough(), setGroove()

Definition at line 860 of file qwt_slider.cpp.

14.144.4.28 setUpdateInterval() `void QwtSlider::setUpdateInterval ( int interval )`

Specify the update interval for automatic scrolling.

The minimal accepted value is 50 ms.

Parameters

interval	Update interval in milliseconds

See also

setUpdateInterval()

Definition at line 395 of file qwt_slider.cpp.

14.144.4.29 sizeHint() `QSize QwtSlider::sizeHint ( ) const [override], [virtual]`

Returns

minimumSizeHint()

Definition at line 912 of file qwt_slider.cpp.

14.144.4.30 sliderRect() `QRect QwtSlider::sliderRect ( ) const [protected]`

Returns

Bounding rectangle of the slider - without the scale

Definition at line 1014 of file qwt_slider.cpp.

14.144.4.31 spacing() `int QwtSlider::spacing ( ) const`

Returns

Number of pixels between slider and scale

See also

setSpacing()

Definition at line 296 of file qwt_slider.cpp.

14.144.4.32 timerEvent() `void QwtSlider::timerEvent ( QTimerEvent * event ) [override], [protected], [virtual]`

Timer event handler

Handles the timer, when the mouse stays pressed inside the sliderRect().

Parameters

event	Mouse event

Definition at line 633 of file qwt_slider.cpp.

14.144.4.33 updateInterval() `int QwtSlider::updateInterval ( ) const`

Returns

Update interval in milliseconds for automatic scrolling

See also

setUpdateInterval()

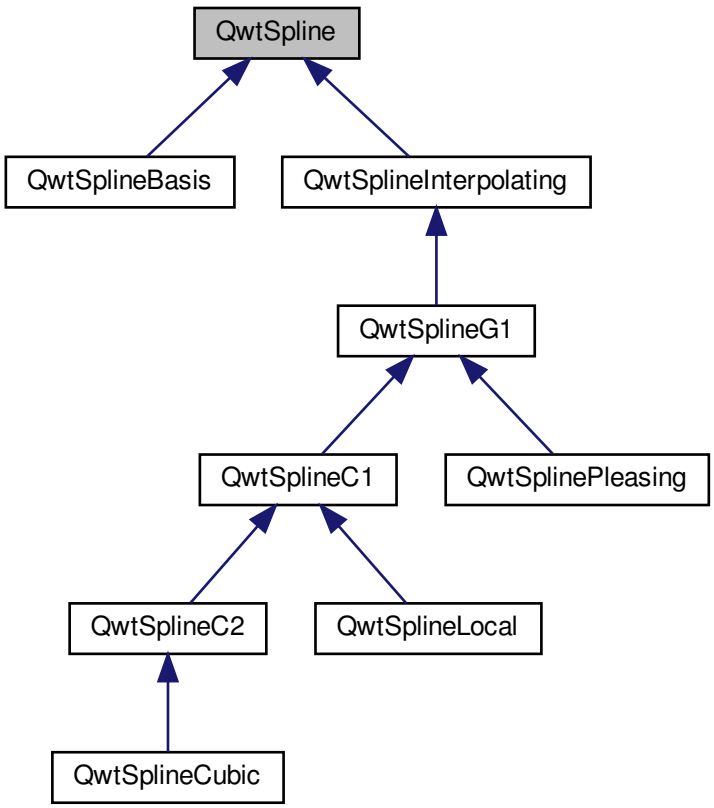
Definition at line 404 of file qwt_slider.cpp.

14.145 QwtSpline Class Reference

Base class for all splines.

`#include <qwt_spline.h>`

Inheritance diagram for QwtSpline:



Public Types

- enum BoundaryType { ConditionalBoundaries , PeriodicPolygon , ClosedPolygon }
- enum BoundaryPosition { AtBeginning , AtEnd }
- enum BoundaryCondition { Clamped1 , Clamped2 , Clamped3 , LinearRunout }

  Boundary condition.

Public Member Functions

- QwtSpline ()

  Constructor.
- virtual ~QwtSpline ()

  Destructor.
- void setParametrization (int type)
- void setParametrization (QwtSplineParametrization *)
- const QwtSplineParametrization * parametrization () const
- void setBoundaryType (BoundaryType)
- BoundaryType boundaryType () const
- void setBoundaryValue (BoundaryPosition, double value)

  Define the boundary value.
- double boundaryValue (BoundaryPosition) const
- void setBoundaryCondition (BoundaryPosition, int condition)

  Define the condition for an endpoint of the spline.
- int boundaryCondition (BoundaryPosition) const
- void setBoundaryConditions (int condition, double valueBegin=0.0, double valueEnd=0.0)

  Define the condition at the endpoints of a spline.
- virtual QPolygonF polygon (const QPolygonF &, double tolerance) const

  Interpolate a curve by a polygon.
- virtual QPainterPath painterPath (const QPolygonF &) const =0
- virtual uint locality () const

14.145.1 Detailed Description

Base class for all splines.

A spline is a curve represented by a sequence of polynomials. Spline approximation is the process of finding polynomials for a given set of points. When the algorithm preserves the initial points it is called interpolating.

Splines can be classified according to conditions of the polynomials that are met at the start/endpoints of the pieces:

- Geometric Continuity
  - G0: polynomials are joined
  - G1: first derivatives are proportional at the join point The curve tangents thus have the same direction, but not necessarily the same magnitude. i.e., C1'(1) = (a,b,c) and C2'(0) = (k*a, k*b, k*c).
  - G2: first and second derivatives are proportional at join point
- Parametric Continuity
  - C0: curves are joined
  - C1: first derivatives equal
  - C2: first and second derivatives are equal

Geometric continuity requires the geometry to be continuous, while parametric continuity requires that the underlying parameterization be continuous as well. Parametric continuity of order n implies geometric continuity of order n, but not vice-versa.

QwtSpline is the base class for spline approximations of any continuity.

Definition at line 57 of file qwt_spline.h.

14.145.2 Member Enumeration Documentation

14.145.2.1 BoundaryCondition `enum QwtSpline::BoundaryCondition`

Boundary condition.

A spline algorithm calculates polynomials by looking a couple of points back/ahead (locality()). At the ends additional rules are necessary to compensate the missing points.

See also

boundaryCondition(), boundaryValue()

QwtSplineC2::BoundaryConditionC2

Enumerator

Clamped1	The first derivative at the end point is given **See also** boundaryValue()
Clamped2	The second derivative at the end point is given **See also** boundaryValue() **Note** a condition having a second derivative of 0 is also called "natural".
Clamped3	The third derivative at the end point is given **See also** boundaryValue() **Note** a condition having a third derivative of 0 is also called "parabolic runout".
LinearRunout	The first derivate at the endpoint is related to the first derivative at its neighbour by the boundary value. F,e when the boundary value at the end is 1.0 then the slope at the last 2 points is the same. **See also** boundaryValue().

Definition at line 119 of file qwt_spline.h.

14.145.2.2 BoundaryPosition `enum QwtSpline::BoundaryPosition`

position of a boundary condition

See also

boundaryCondition(), boundaryValue()

Enumerator

AtBeginning	the condition is at the beginning of the polynomial
AtEnd	the condition is at the end of the polynomial

Definition at line 99 of file qwt_spline.h.

14.145.2.3 BoundaryType `enum QwtSpline::BoundaryType`

Boundary type specifying the spline at its endpoints

See also

setBoundaryType(), boundaryType()

Enumerator

ConditionalBoundaries	The polynomials at the start/endpoint depend on specific conditions **See also** QwtSpline::BoundaryCondition
PeriodicPolygon	The polynomials at the start/endpoint are found by using imaginary additional points. Additional points at the end are found by translating points from the beginning or v.v.
ClosedPolygon	ClosedPolygon is similar to PeriodicPolygon beside, that the interpolation includes the connection between the last and the first control point. **Note** Only works for parametrizations, where the parameter increment for the the final closing line is positive. This excludes QwtSplineParametrization::ParameterX and QwtSplineParametrization::ParameterY

Definition at line 65 of file qwt_spline.h.

14.145.3 Constructor & Destructor Documentation

14.145.3.1 QwtSpline() `QwtSpline::QwtSpline ( )`

Constructor.

The default setting is a non closing spline with chordal parametrization

See also

setParametrization(), setBoundaryType()

Definition at line 540 of file qwt_spline.cpp.

14.145.4 Member Function Documentation

14.145.4.1 boundaryCondition() `int QwtSpline::boundaryCondition ( BoundaryPosition position ) const`

Returns

Condition for an endpoint of the spline

Parameters

position	At the beginning or the end of the spline

See also

setBoundaryCondition(), boundaryValue(), setBoundaryConditions()

Definition at line 651 of file qwt_spline.cpp.

14.145.4.2 boundaryType() `QwtSpline::BoundaryType QwtSpline::boundaryType ( ) const`

Returns

Boundary type

See also

setBoundaryType()

Definition at line 626 of file qwt_spline.cpp.

14.145.4.3 boundaryValue() double QwtSpline::boundaryValue (BoundaryPosition *position*) const

Returns

Boundary value

Parameters

position	At the beginning or the end of the spline

See also

setBoundaryValue(), boundaryCondition()

Definition at line 682 of file qwt_spline.cpp.

14.145.4.4 locality() `uint QwtSpline::locality ( ) const [virtual]`

The locality of an spline interpolation identifies how many adjacent polynomials are affected, when changing the position of one point.

A locality of 'n' means, that changing the coordinates of a point has an effect on 'n' leading and 'n' following polynomials. Those polynomials can be calculated from a local subpolygon.

A value of 0 means, that the interpolation is not local and any modification of the polygon requires to recalculate all polynomials (f.e cubic splines).

Returns

Order of locality

Reimplemented in QwtSplinePleasing, QwtSplineLocal, QwtSplineCubic, and QwtSplineBasis.

Definition at line 564 of file qwt_spline.cpp.

14.145.4.5 painterPath() `QPainterPath QwtSpline::painterPath ( const QPolygonF & points ) const [pure virtual]`

Approximates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

Parameters

points	Control points

Returns

Painter path, that can be rendered by QPainter

See also

polygon(), QwtBezier

Implemented in QwtSplinePleasing, QwtSplineLocal, QwtSplineCubic, QwtSplineBasis, QwtSplineC2, QwtSplineC1, and QwtSplineInterpolating.

14.145.4.6 parametrization() `const QwtSplineParametrization * QwtSpline::parametrization ( ) const`

Returns

parametrization

See also

setParametrization()

Definition at line 605 of file qwt_spline.cpp.

14.145.4.7 polygon()

```
QPolygonF QwtSpline::polygon (
    const QPolygonF & points,
    double tolerance ) const  [virtual]
```

Interpolate a curve by a polygon.

Interpolates a polygon piecewise with Bezier curves interpolating them in a 2nd pass by polygons.

The interpolation is based on "Piecewise Linear Approximation of Bézier Curves" by Roger Willcocks (http://www.rops.org)

Parameters

points	Control points
tolerance	Maximum for the accepted error of the approximation

Returns

polygon approximating the interpolating polynomials

See also

bezierControlLines(), QwtBezier

Reimplemented in QwtSplineInterpolating.

Definition at line 496 of file qwt_spline.cpp.

14.145.4.8 setBoundaryCondition()

```
void QwtSpline::setBoundaryCondition (
    BoundaryPosition position,
    int condition )
```

Define the condition for an endpoint of the spline.

Parameters

position	At the beginning or the end of the spline
condition	Condition

See also

BoundaryCondition, QwtSplineC2::BoundaryCondition, boundaryCondition()

Definition at line 639 of file qwt_spline.cpp.

14.145.4.9 setBoundaryConditions()

```
void QwtSpline::setBoundaryConditions (
    int condition,
    double valueBegin = 0.0,
    double valueEnd = 0.0 )
```

Define the condition at the endpoints of a spline.

Parameters

condition	Condition
valueBegin	Used for the condition at the beginning of te spline
valueEnd	Used for the condition at the end of te spline

See also

BoundaryCondition, QwtSplineC2::BoundaryCondition, testBoundaryCondition(), setBoundaryValue()

Definition at line 700 of file qwt_spline.cpp.

14.145.4.10 setBoundaryType()

```
void QwtSpline::setBoundaryType (
    BoundaryType boundaryType )
```

Define the boundary type for the endpoints of the approximating spline.

Parameters

boundaryType	Boundary type

See also

boundaryType()

Definition at line 617 of file qwt_spline.cpp.

14.145.4.11 setBoundaryValue() `void QwtSpline::setBoundaryValue (`

```
BoundaryPosition position,
double value )
```

Define the boundary value.

The boundary value is an parameter used in combination with the boundary condition. Its meaning depends on the condition.

Parameters

position	At the beginning or the end of the spline
value	Value used for the condition at the end point

See also

boundaryValue(), setBoundaryCondition()

Definition at line 670 of file qwt_spline.cpp.

14.145.4.12 setParametrization() [1/2] `void QwtSpline::setParametrization (`

```
int type )
```

Define the parametrization for a parametric spline approximation The default setting is a chordal parametrization.

Parameters

type	Type of parametrization, usually one of QwtSplineParametrization::Type

See also

parametrization()

Definition at line 576 of file qwt_spline.cpp.

14.145.4.13 setParametrization() [2/2] `void QwtSpline::setParametrization (`

```
QwtSplineParametrization * parametrization )
```

Define the parametrization for a parametric spline approximation The default setting is a chordal parametrization.

Parameters

parametrization	Parametrization

See also

parametrization()

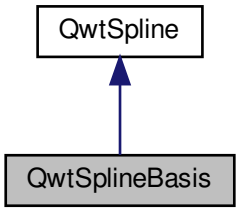
Definition at line 592 of file qwt_spline.cpp.

14.146 QwtSplineBasis Class Reference

An approximation using a basis spline.

`#include <qwt_spline_basis.h>`

Inheritance diagram for QwtSplineBasis:



Public Member Functions

- QwtSplineBasis ()
  Constructor.
- virtual ~QwtSplineBasis ()
  Destructor.
- virtual QPainterPath painterPath (const QPolygonF &) const override
- virtual uint locality () const override
  The locality is always 2.

Additional Inherited Members

14.146.1 Detailed Description

An approximation using a basis spline.

QwtSplineBasis approximates a set of points by a polynomials with C2 continuity (= first and second derivatives are equal) at the end points.

The end points of the spline do not match the original points.

Definition at line 24 of file qwt_spline_basis.h.

14.146.2 Member Function Documentation

14.146.2.1 painterPath() `QPainterPath QwtSplineBasis::painterPath ( const QPolygonF & points ) const [override], [virtual]`

Approximates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

Parameters

points	Control points

Returns

Painter path, that can be rendered by QPainter

Implements QwtSpline.

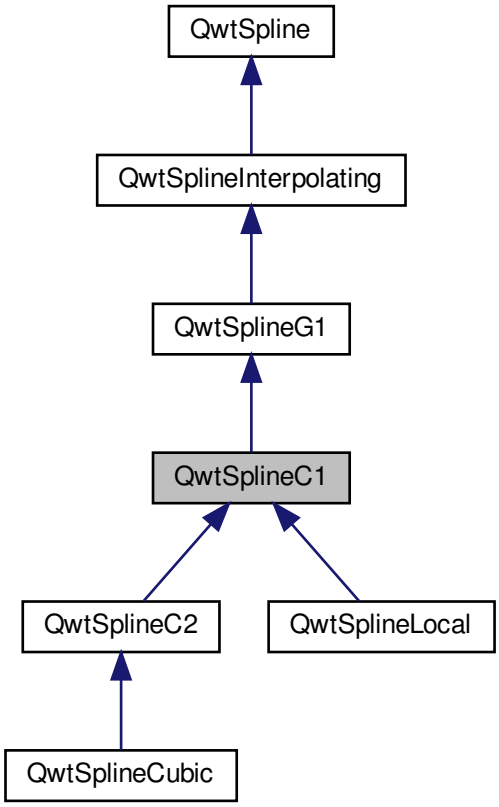
Definition at line 249 of file qwt_spline_basis.cpp.

14.147 QwtSplineC1 Class Reference

Base class for spline interpolations providing a first order parametric continuity (C1) between adjoining curves.

`#include <qwt_spline.h>`

Inheritance diagram for QwtSplineC1:



Public Member Functions

- QwtSplineC1 ()

  Constructor.
- virtual ~QwtSplineC1 ()

  Destructor.
- virtual QPainterPath painterPath (const QPolygonF &) const override

  Calculate an interpolated painter path.
- virtual QVector< QLineF > bezierControlLines (const QPolygonF &) const override

  Interpolate a curve with Bezier curves.
- virtual QPolygonF equidistantPolygon (const QPolygonF &, double distance, bool withNodes) const override

  Find an interpolated polygon with "equidistant" points.
- virtual QVector< QwtSplinePolynomial > polynomials (const QPolygonF &) const

  Calculate the interpolating polynomials for a non parametric spline.
- virtual QVector< double > slopes (const QPolygonF &) const =0

  Find the first derivative at the control points.
- virtual double slopeAtBeginning (const QPolygonF &, double slopeNext) const
- virtual double slopeAtEnd (const QPolygonF &, double slopeBefore) const

Additional Inherited Members

14.147.1 Detailed Description

Base class for spline interpolations providing a first order parametric continuity (C1) between adjoining curves.

All interpolations with C1 continuity are based on rules for finding the 1. derivate at some control points.

In case of non parametric splines those points are the curve points, while for parametric splines the calculation is done twice using a parameter value t.

See also

QwtSplineParametrization

Definition at line 235 of file qwt_spline.h.

14.147.2 Constructor & Destructor Documentation

14.147.2.1 QwtSplineC1() `QwtSplineC1::QwtSplineC1 ( )`

Constructor.

The default setting is a non closing spline with no parametrization (QwtSplineParametrization::ParameterX).

See also

QwtSpline::setParametrization(), QwtSpline::setBoundaryType()

Definition at line 962 of file qwt_spline.cpp.

14.147.3 Member Function Documentation

14.147.3.1 bezierControlLines() QVector< QLineF > QwtSplineC1::bezierControlLines (const QPolygonF & *points*) const [override], [virtual]

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns the 2 control points of each curve as QLineF.

Parameters

points	Control points

Returns

Control points of the interpolating Bezier curves

Implements QwtSplineInterpolating.

Reimplemented in QwtSplineCubic, QwtSplineLocal, and QwtSplineC2.

Definition at line 1101 of file qwt_spline.cpp.

14.147.3.2 equidistantPolygon() QPolygonF QwtSplineC1::equidistantPolygon (const QPolygonF & *points*, double *distance*, bool *withNodes*) const [override], [virtual]

Find an interpolated polygon with "equidistant" points.

The implementation is optimzed for non parametric curves (QwtSplineParametrization::ParameterX) and falls back to QwtSpline::equidistantPolygon() otherwise.

Parameters

points	Control nodes of the spline
distance	Distance between 2 points according to the parametrization
withNodes	When true, also add the control nodes (even if not being equidistant)

Returns

Interpolating polygon

See also

QwtSpline::equidistantPolygon()

Reimplemented from QwtSplineInterpolating.

Reimplemented in QwtSplineC2.

Definition at line 1167 of file qwt_spline.cpp.

14.147.3.3 painterPath()

```
QPainterPath QwtSplineC1::painterPath (
        const QPolygonF & points ) const  [override], [virtual]
```

Calculate an interpolated painter path.

Interpolates a polygon piecewise into cubic Bezier curves and returns them as QPainterPath.

The implementation calculates the slopes at the control points and converts them into painter path elements in an additional loop.

Parameters

points	Control points

Returns

QPainterPath Painter path, that can be rendered by QPainter

Note

Derived spline classes might overload painterPath() to avoid the extra loops for converting results into a QPainterPath

Reimplemented from QwtSplineInterpolating.

Reimplemented in QwtSplineLocal, QwtSplineCubic, and QwtSplineC2.

Definition at line 1043 of file qwt_spline.cpp.

14.147.3.4 polynomials()

```
QVector< QwtSplinePolynomial > QwtSplineC1::polynomials (
        const QPolygonF & points ) const  [virtual]
```

Calculate the interpolating polynomials for a non parametric spline.

C1 spline interpolations are based on finding values for the first derivates at the control points. The interpolating polynomials can be calculated from the the first derivates using QwtSplinePolynomial::fromSlopes().

The default implementation is a two pass calculation. In derived classes it might be overloaded by a one pass implementation.

Parameters

points	Control points

Returns

Interpolating polynomials

Note

The x coordinates need to be increasing or decreasing

Reimplemented in QwtSplineLocal, QwtSplineCubic, and QwtSplineC2.

Definition at line 1201 of file qwt_spline.cpp.

14.147.3.5 slopeAtBeginning() `double QwtSplineC1::slopeAtBeginning ( const QPolygonF & points, double slopeNext ) const [virtual]`

Parameters

points	Control points
slopeNext	Value of the first derivative at the second point

Returns

value of the first derivative at the first point

See also

slopeAtEnd(), QwtSpline::boundaryCondition(), QwtSpline::boundaryValue()

Definition at line 979 of file qwt_spline.cpp.

14.147.3.6 slopeAtEnd() `double QwtSplineC1::slopeAtEnd ( const QPolygonF & points, double slopeBefore ) const [virtual]`

Parameters

points	Control points
slopeBefore	Value of the first derivative at the point before the last one

Returns

value of the first derivative at the last point

See also

slopeAtBeginning(), QwtSpline::boundaryCondition(), QwtSpline::boundaryValue()

Definition at line 997 of file qwt_spline.cpp.

14.147.3.7 slopes()

```
QVector< double > QwtSplineC1::slopes (
        const QPolygonF & points ) const  [pure virtual]
```

Find the first derivative at the control points.

Parameters

points	Control nodes of the spline

Returns

Vector with the values of the 2nd derivate at the control points

Note

The x coordinates need to be increasing or decreasing

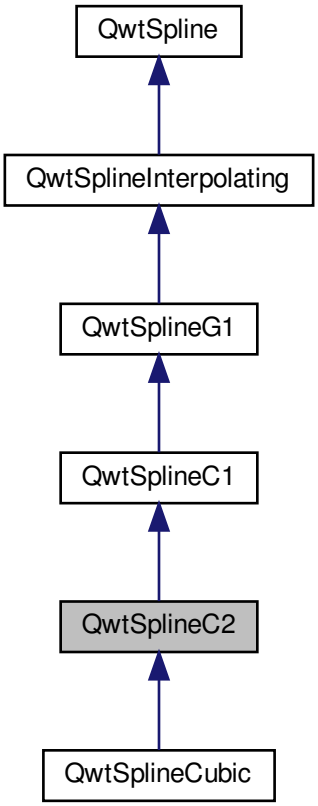
Implemented in QwtSplineLocal, QwtSplineCubic, and QwtSplineC2.

14.148 QwtSplineC2 Class Reference

Base class for spline interpolations providing a second order parametric continuity (C2) between adjoining curves.

```
#include <qwt_spline.h>
```

Inheritance diagram for QwtSplineC2:



Public Types

- enum BoundaryConditionC2 { CubicRunout = LinearRunout + 1 , NotAKnot }

Public Member Functions

- QwtSplineC2 ()

  Constructor.
- virtual ~QwtSplineC2 ()

  Destructor.
- virtual QPainterPath painterPath (const QPolygonF &) const override

  Interpolate a curve with Bezier curves.
- virtual QVector< QLineF > bezierControlLines (const QPolygonF &) const override

  Interpolate a curve with Bezier curves.
- virtual QPolygonF equidistantPolygon (const QPolygonF &, double distance, bool withNodes) const override

  Find an interpolated polygon with "equidistant" points.
- virtual QVector< QwtSplinePolynomial > polynomials (const QPolygonF &) const override

  Calculate the interpolating polynomials for a non parametric spline.

- virtual QVector< double > slopes (const QPolygonF &) const override

  Find the first derivative at the control points.
- virtual QVector< double > curvatures (const QPolygonF &) const =0

  Find the second derivative at the control points.

14.148.1 Detailed Description

Base class for spline interpolations providing a second order parametric continuity (C2) between adjoining curves.

All interpolations with C2 continuity are based on rules for finding the 2. derivate at some control points.

In case of non parametric splines those points are the curve points, while for parametric splines the calculation is done twice using a parameter value t.

See also

QwtSplineParametrization

Definition at line 267 of file qwt_spline.h.

14.148.2 Member Enumeration Documentation

14.148.2.1 BoundaryConditionC2 enum QwtSplineC2::BoundaryConditionC2

Boundary condition that requires C2 continuity

See also

QwtSpline::boundaryCondition, QwtSpline::BoundaryCondition

Enumerator

CubicRunout	The second derivate at the endpoint is related to the second derivatives at the 2 neighbours: cv[0] := 2.0 * cv[1] - cv[2]. **Note** boundaryValue() is ignored
NotAKnot	The 3rd derivate at the endpoint matches the 3rd derivate at its neighbours. Or in other words: the first/last curve segment extents the polynomial of its neighboured polynomial **Note** boundaryValue() is ignored

Definition at line 275 of file qwt_spline.h.

14.148.3 Constructor & Destructor Documentation

14.148.3.1 QwtSplineC2() `QwtSplineC2::QwtSplineC2 ( )`

Constructor.

The default setting is a non closing spline with no parametrization (QwtSplineParametrization::ParameterX).

See also

QwtSpline::setParametrization(), QwtSpline::setBoundaryType()

Definition at line 1228 of file qwt_spline.cpp.

14.148.4 Member Function Documentation

14.148.4.1 bezierControlLines() `QVector< QLineF > QwtSplineC2::bezierControlLines ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns the 2 control points of each curve as QLineF.

Parameters

points	Control points

Returns

Control points of the interpolating Bezier curves

Note

The implementation simply calls QwtSplineC1::bezierControlLines(), but is intended to be replaced by a more efficient implementation that builds the polynomials by the curvatures some day.

Reimplemented from QwtSplineC1.

Reimplemented in QwtSplineCubic.

Definition at line 1270 of file qwt_spline.cpp.

14.148.4.2 curvatures() `QVector< double > QwtSplineC2::curvatures ( const QPolygonF & points ) const [pure virtual]`

Find the second derivative at the control points.

Parameters

points	Control nodes of the spline

Returns

Vector with the values of the 2nd derivate at the control points

See also

slopes()

Note

The x coordinates need to be increasing or decreasing

Implemented in QwtSplineCubic.

14.148.4.3 equidistantPolygon()

```
QPolygonF QwtSplineC2::equidistantPolygon (
    const QPolygonF & points,
    double distance,
    bool withNodes ) const  [override], [virtual]
```

Find an interpolated polygon with "equidistant" points.

The implementation is optimzed for non parametric curves (QwtSplineParametrization::ParameterX) and falls back to QwtSpline::equidistantPolygon() otherwise.

Parameters

points	Control nodes of the spline
distance	Distance between 2 points according to the parametrization
withNodes	When true, also add the control nodes (even if not being equidistant)

Returns

Interpolating polygon

See also

QwtSpline::equidistantPolygon()

Reimplemented from QwtSplineC1.

Definition at line 1295 of file qwt_spline.cpp.

14.148.4.4 painterPath() `QPainterPath QwtSplineC2::painterPath ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

Parameters

points	Control points

Returns

Painter path, that can be rendered by QPainter

Note

The implementation simply calls QwtSplineC1::painterPath(), but is intended to be replaced by a one pass calculation some day.

Reimplemented from QwtSplineC1.

Reimplemented in QwtSplineCubic.

Definition at line 1249 of file qwt_spline.cpp.

14.148.4.5 polynomials() `QVector< QwtSplinePolynomial > QwtSplineC2::polynomials ( const QPolygonF & points ) const [override], [virtual]`

Calculate the interpolating polynomials for a non parametric spline.

C2 spline interpolations are based on finding values for the second derivates of f at the control points. The interpolating polynomials can be calculated from the the second derivates using QwtSplinePolynomial::fromCurvatures.

The default implementation is a 2 pass calculation. In derived classes it might be overloaded by a one pass implementation.

Parameters

points	Control points

Returns

Interpolating polynomials

Note

The x coordinates need to be increasing or decreasing

Reimplemented from QwtSplineC1.

Reimplemented in QwtSplineCubic.

Definition at line 1381 of file qwt_spline.cpp.

14.148.4.6 slopes() QVector< double > QwtSplineC2::slopes (const QPolygonF & *points*) const [override], [virtual]

Find the first derivative at the control points.

An implementation calculating the 2nd derivatives and then building the slopes in a 2nd loop. QwtSplineCubic overloads it with a more performant implementation doing it in one loop.

Parameters

points	Control nodes of the spline

Returns

Vector with the values of the 1nd derivate at the control points

See also

curvatures()

Note

The x coordinates need to be increasing or decreasing

Implements QwtSplineC1.

Reimplemented in QwtSplineCubic.

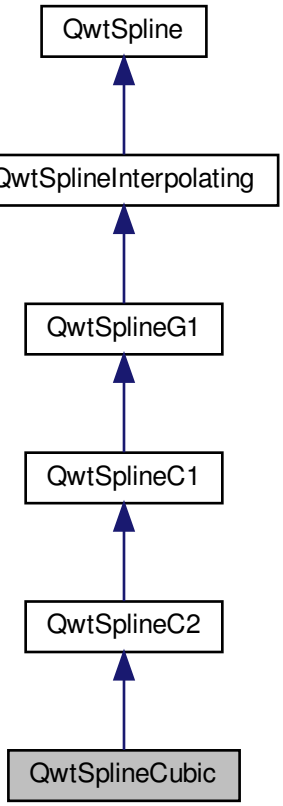
Definition at line 1339 of file qwt_spline.cpp.

14.149 QwtSplineCubic Class Reference

A cubic spline.

```
#include <qwt_spline_cubic.h>
```

Inheritance diagram for QwtSplineCubic:



Public Member Functions

- QwtSplineCubic ()

  Constructor The default setting is a non closing natural spline with no parametrization.
- virtual ~QwtSplineCubic ()

  Destructor.
- virtual uint locality () const override
- virtual QPainterPath painterPath (const QPolygonF &) const override

  Interpolate a curve with Bezier curves.
- virtual QVector< QLineF > bezierControlLines (const QPolygonF &points) const override

  Interpolate a curve with Bezier curves.
- virtual QVector< QwtSplinePolynomial > polynomials (const QPolygonF &) const override

  Calculate the interpolating polynomials for a non parametric spline.
- virtual QVector< double > slopes (const QPolygonF &) const override

  Find the first derivative at the control points.
- virtual QVector< double > curvatures (const QPolygonF &) const override

  Find the second derivative at the control points.

Additional Inherited Members

14.149.1 Detailed Description

A cubic spline.

A cubic spline is a spline with C2 continuity at all control points. It is a non local spline, what means that all polynomials are changing when one control point has changed.

The implementation is based on the fact, that the continuity condition means an equation with 3 unknowns for 3 adjacent points. The equation system can be resolved by defining start/end conditions, that allow substituting of one of the unknowns for the start/end equations.

Resolving the equation system is a 2 pass algorithm, requiring more CPU costs than all other implemented type of splines.

Definition at line 33 of file qwt_spline_cubic.h.

14.149.2 Member Function Documentation

14.149.2.1 bezierControlLines() `QVector< QLineF > QwtSplineCubic::bezierControlLines ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns the 2 control points of each curve as QLineF.

Parameters

points	Control points

Returns

Control points of the interpolating Bezier curves

Note

The implementation simply calls QwtSplineC1::bezierControlLines()

Reimplemented from QwtSplineC2.

Definition at line 1149 of file qwt_spline_cubic.cpp.

14.149.2.2 curvatures() `QVector< double > QwtSplineCubic::curvatures ( const QPolygonF & points ) const [override], [virtual]`

Find the second derivative at the control points.

Parameters

points	Control nodes of the spline

Returns

Vector with the values of the 2nd derivate at the control points

See also

slopes()

Note

The x coordinates need to be increasing or decreasing

Implements QwtSplineC2.

Definition at line 1078 of file qwt_spline_cubic.cpp.

14.149.2.3 locality() `uint QwtSplineCubic::locality ( ) const [override], [virtual]`

A cubic spline is non local, where changing one point has em effect on all polynomials.

Returns

0

Reimplemented from QwtSpline.

Definition at line 989 of file qwt_spline_cubic.cpp.

14.149.2.4 painterPath() `QPainterPath QwtSplineCubic::painterPath ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

Parameters

points	Control points

Returns

Painter path, that can be rendered by QPainter

Note

The implementation simply calls QwtSplineC1::painterPath()

Reimplemented from QwtSplineC2.

Definition at line 1130 of file qwt_spline_cubic.cpp.

14.149.2.5 polynomials()

```
QVector< QwtSplinePolynomial > QwtSplineCubic::polynomials (
    const QPolygonF & points ) const  [override], [virtual]
```

Calculate the interpolating polynomials for a non parametric spline.

Parameters

points	Control points

Returns

Interpolating polynomials

Note

The x coordinates need to be increasing or decreasing

The implementation simply calls QwtSplineC2::polynomials(), but is intended to be replaced by a one pass calculation some day.

Reimplemented from QwtSplineC2.

Definition at line 1167 of file qwt_spline_cubic.cpp.

14.149.2.6 slopes()

```
QVector< double > QwtSplineCubic::slopes (
    const QPolygonF & points ) const  [override], [virtual]
```

Find the first derivative at the control points.

In opposite to the implementation QwtSplineC2::slopes the first derivates are calculated directly, without calculating the second derivates first.

Parameters

points	Control nodes of the spline

Returns

Vector with the values of the 2nd derivate at the control points

See also

curvatures(), QwtSplinePolynomial::fromCurvatures()

Note

The x coordinates need to be increasing or decreasing

Reimplemented from QwtSplineC2.

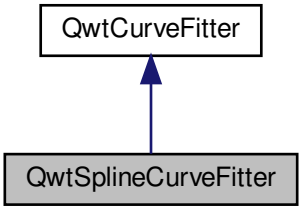
Definition at line 1006 of file qwt_spline_cubic.cpp.

14.150 QwtSplineCurveFitter Class Reference

A curve fitter using a spline interpolation.

`#include <qwt_spline_curve_fitter.h>`

Inheritance diagram for QwtSplineCurveFitter:



Public Member Functions

- QwtSplineCurveFitter ()
  Constructor.
- virtual ~QwtSplineCurveFitter ()
  Destructor.
- void setSpline (QwtSpline *)
- const QwtSpline * spline () const
- QwtSpline * spline ()
- virtual QPolygonF fitCurve (const QPolygonF &) const override
- virtual QPainterPath fitCurvePath (const QPolygonF &) const override

Additional Inherited Members

14.150.1 Detailed Description

A curve fitter using a spline interpolation.

The default setting for the spline is a cardinal spline with uniform parametrization.

See also

QwtSpline, QwtSplineLocal

Definition at line 25 of file qwt_spline_curve_fitter.h.

14.150.2 Member Function Documentation

14.150.2.1 fitCurve() `QPolygonF QwtSplineCurveFitter::fitCurve ( const QPolygonF & points ) const [override], [virtual]`

Find a curve which has the best fit to a series of data points

Parameters

points	Series of data points

Returns

Fitted Curve

See also

fitCurvePath()

Implements QwtCurveFitter.

Definition at line 75 of file qwt_spline_curve_fitter.cpp.

14.150.2.2 fitCurvePath() `QPainterPath QwtSplineCurveFitter::fitCurvePath ( const QPolygonF & points ) const [override], [virtual]`

Find a curve path which has the best fit to a series of data points

Parameters

points	Series of data points

Returns

Fitted Curve

See also

fitCurve()

Implements QwtCurveFitter.

Definition at line 94 of file qwt_spline_curve_fitter.cpp.

14.150.2.3 setSpline() `void QwtSplineCurveFitter::setSpline ( QwtSpline * spline )`

Assign a spline

The spline needs to be allocated by new and will be deleted in the destructor of the fitter.

Parameters

spline	Spline

See also

spline()

Definition at line 40 of file qwt_spline_curve_fitter.cpp.

14.150.2.4 spline() [1/2] `QwtSpline * QwtSplineCurveFitter::spline ( )`

Returns

Spline

See also

setSpline()

Definition at line 62 of file qwt_spline_curve_fitter.cpp.

14.150.2.5 spline() [2/2] const QwtSpline * QwtSplineCurveFitter::spline ( ) const

Returns

Spline

See also

setSpline()

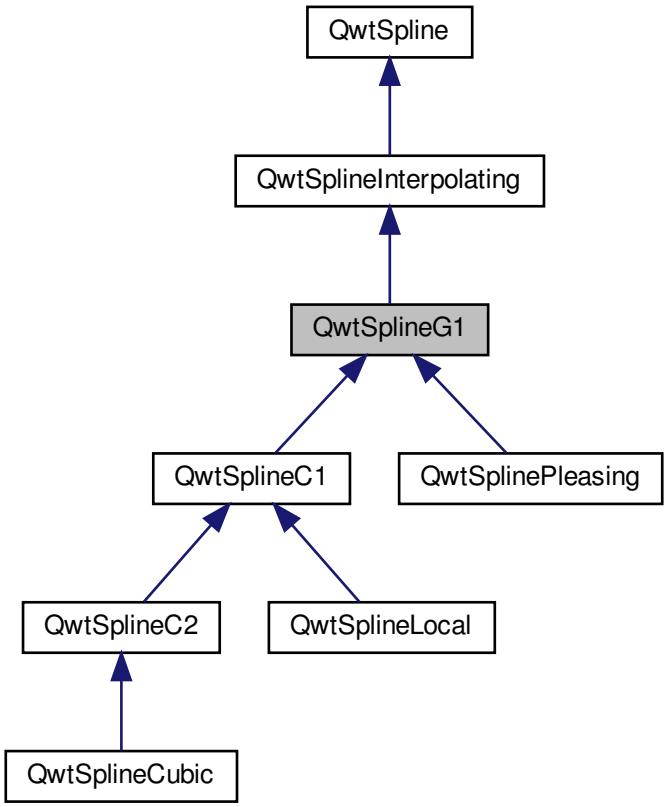
Definition at line 53 of file qwt_spline_curve_fitter.cpp.

14.151 QwtSplineG1 Class Reference

Base class for spline interpolations providing a first order geometric continuity (G1) between adjoining curves.

`#include <qwt_spline.h>`

Inheritance diagram for QwtSplineG1:



Public Member Functions

- QwtSplineG1 ()
  Constructor.
- virtual ~QwtSplineG1 ()
  Destructor.

Additional Inherited Members

14.151.1 Detailed Description

Base class for spline interpolations providing a first order geometric continuity (G1) between adjoining curves.

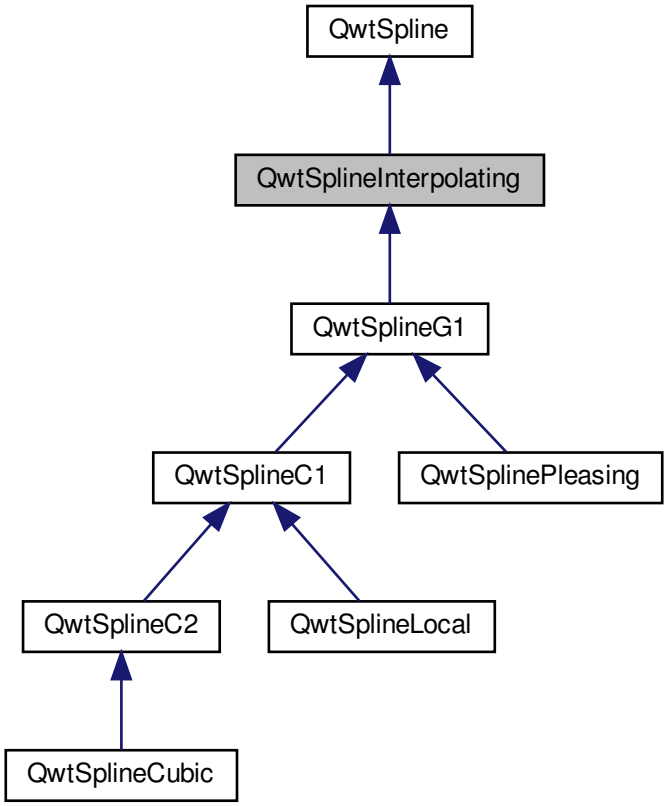
Definition at line 216 of file qwt_spline.h.

14.152 QwtSplineInterpolating Class Reference

Base class for a spline interpolation.

`#include <qwt_spline.h>`

Inheritance diagram for QwtSplineInterpolating:



Public Member Functions

- QwtSplineInterpolating ()

  Constructor.
- virtual ~QwtSplineInterpolating ()

  Destructor.
- virtual QPolygonF equidistantPolygon (const QPolygonF &, double distance, bool withNodes) const

  Find an interpolated polygon with "equidistant" points.
- virtual QPolygonF polygon (const QPolygonF &, double tolerance) const override

  Interpolate a curve by a polygon.
- virtual QPainterPath painterPath (const QPolygonF &) const override

  Interpolate a curve with Bezier curves.
- virtual QVector< QLineF > bezierControlLines (const QPolygonF &) const =0

  Interpolate a curve with Bezier curves.

Additional Inherited Members

14.152.1 Detailed Description

Base class for a spline interpolation.

Spline interpolation is the process of interpolating a set of points piecewise with polynomials. The initial set of points is preserved.

Definition at line 193 of file qwt_spline.h.

14.152.2 Member Function Documentation

14.152.2.1 bezierControlLines() `QVector< QLineF > QwtSplineInterpolating::bezierControlLines ( const QPolygonF & points ) const [pure virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns the 2 control points of each curve as QLineF.

Parameters

points	Control points

Returns

Control points of the interpolating Bezier curves

Implemented in QwtSplineCubic, QwtSplinePleasing, QwtSplineLocal, QwtSplineC2, and QwtSplineC1.

14.152.2.2 equidistantPolygon()

```
QPolygonF QwtSplineInterpolating::equidistantPolygon (
    const QPolygonF & points,
    double distance,
    bool withNodes ) const  [virtual]
```

Find an interpolated polygon with "equidistant" points.

When withNodes is disabled all points of the resulting polygon will be equidistant according to the parametrization.

When withNodes is enabled the resulting polygon will also include the control points and the interpolated points are always aligned to the control point before (points[i] + i ∗ distance).

The implementation calculates bezier curves first and calculates the interpolated points in a second run.

Parameters

points	Control nodes of the spline
distance	Distance between 2 points according to the parametrization
withNodes	When true, also add the control nodes (even if not being equidistant)

Returns

Interpolating polygon

See also

bezierControlLines()

Reimplemented in QwtSplineC2, and QwtSplineC1.

Definition at line 863 of file qwt_spline.cpp.

14.152.2.3 painterPath()

```
QPainterPath QwtSplineInterpolating::painterPath (
    const QPolygonF & points ) const  [override], [virtual]
```

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

The implementation calculates the Bezier control lines first and converts them into painter path elements in an additional loop.

Parameters

points	Control points

Returns

Painter path, that can be rendered by QPainter

Note

Derived spline classes might overload painterPath() to avoid the extra loops for converting results into a QPainterPath

See also

bezierControlLines()

Implements QwtSpline.

Reimplemented in QwtSplinePleasing, QwtSplineLocal, QwtSplineCubic, QwtSplineC2, and QwtSplineC1.

Definition at line 748 of file qwt_spline.cpp.

14.152.2.4 polygon()

```
QPolygonF QwtSplineInterpolating::polygon (
    const QPolygonF & points,
    double tolerance ) const  [override], [virtual]
```

Interpolate a curve by a polygon.

Interpolates a polygon piecewise with Bezier curves approximating them by polygons.

The approximation is based on "Piecewise Linear Approximation of Bézier Curves" by Roger Willcocks (http://www.rops.org)

Parameters

points	Control points
tolerance	Maximum for the accepted error of the approximation

Returns

polygon approximating the interpolating polynomials

See also

bezierControlLines(), QwtSplineBezier::toPolygon()

Reimplemented from QwtSpline.

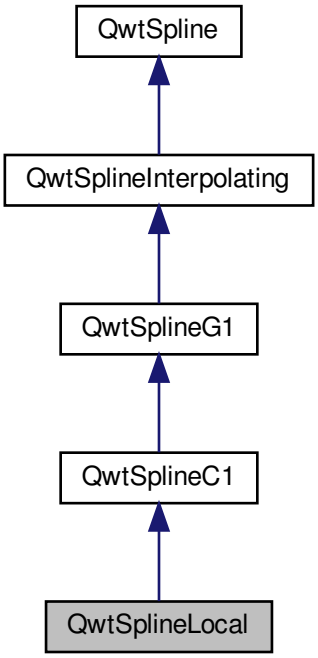
Definition at line 805 of file qwt_spline.cpp.

14.153 QwtSplineLocal Class Reference

A spline with C1 continuity.

```
#include <qwt_spline_local.h>
```

Inheritance diagram for QwtSplineLocal:



Public Types

- enum Type { Cardinal , ParabolicBlending , Akima , PChip }
  Spline interpolation type.

Public Member Functions

- QwtSplineLocal (Type type)
  Constructor.
- virtual ~QwtSplineLocal ()
  Destructor.
- Type type () const
- virtual uint locality () const override
- virtual QPainterPath painterPath (const QPolygonF &) const override
  Interpolate a curve with Bezier curves.
- virtual QVector< QLineF > bezierControlLines (const QPolygonF &) const override
  Interpolate a curve with Bezier curves.
- virtual QVector< QwtSplinePolynomial > polynomials (const QPolygonF &) const override
  Calculate the interpolating polynomials for a non parametric spline.
- virtual QVector< double > slopes (const QPolygonF &) const override
  Find the first derivative at the control points.

14.153.1 Detailed Description

A spline with C1 continuity.

QwtSplineLocal offers several standard algorithms for interpolating a curve with polynomials having C1 continuity at the control points. All algorithms are local in a sense, that changing one control point only few polynomials.

Definition at line 24 of file qwt_spline_local.h.

14.153.2 Member Enumeration Documentation

14.153.2.1 Type `enum QwtSplineLocal::Type`

Spline interpolation type.

All type of spline interpolations are lightweight algorithms calculating the slopes at a point by looking 1 or 2 points back and ahead.

Enumerator

Cardinal	A cardinal spline The cardinal spline interpolation is a very cheap calculation with a locality of 1.
ParabolicBlending	Parabolic blending is a cheap calculation with a locality of 1. Sometimes it is also called Cubic Bessel interpolation.
Akima	The algorithm of H.Akima is a calculation with a locality of 2.
PChip	Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) is an algorithm that is popular because of being offered by MATLAB. It preserves the shape of the data and respects monotonicity. It has a locality of 1.

Definition at line 34 of file qwt_spline_local.h.

14.153.3 Constructor & Destructor Documentation

14.153.3.1 QwtSplineLocal() `QwtSplineLocal::QwtSplineLocal ( Type type )`

Constructor.

Parameters

type	Spline type, specifying the type of interpolation

See also

type()

Definition at line 450 of file qwt_spline_local.cpp.

14.153.4 Member Function Documentation

14.153.4.1 bezierControlLines() `QVector< QLineF > QwtSplineLocal::bezierControlLines ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns the 2 control points of each curve as QLineF.

Parameters

points	Control points

Returns

Control points of the interpolating Bezier curves

Reimplemented from QwtSplineC1.

Definition at line 502 of file qwt_spline_local.cpp.

14.153.4.2 locality() `uint QwtSplineLocal::locality ( ) const [override], [virtual]`

The locality of an spline interpolation identifies how many adjacent polynomials are affected, when changing the position of one point.

The Cardinal, ParabolicBlending and PChip algorithms have a locality of 1, while the Akima interpolation has a locality of 2.

Returns

1 or 2.

Reimplemented from QwtSpline.

Definition at line 552 of file qwt_spline_local.cpp.

14.153.4.3 painterPath() `QPainterPath QwtSplineLocal::painterPath ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

Parameters

points	Control points

Returns

Painter path, that can be rendered by QPainter

Reimplemented from QwtSplineC1.

Definition at line 482 of file qwt_spline_local.cpp.

14.153.4.4 polynomials()

```
QVector< QwtSplinePolynomial > QwtSplineLocal::polynomials (
    const QPolygonF & points ) const  [override], [virtual]
```

Calculate the interpolating polynomials for a non parametric spline.

Parameters

points	Control points

Returns

Interpolating polynomials

Note

The x coordinates need to be increasing or decreasing

The implementation simply calls QwtSplineC1::polynomials(), but is intended to be replaced by a one pass calculation some day.

Reimplemented from QwtSplineC1.

Definition at line 537 of file qwt_spline_local.cpp.

14.153.4.5 slopes()

```
QVector< double > QwtSplineLocal::slopes (
    const QPolygonF & points ) const  [override], [virtual]
```

Find the first derivative at the control points.

Parameters

points	Control nodes of the spline

Returns

Vector with the values of the 2nd derivate at the control points

Note

The x coordinates need to be increasing or decreasing

Implements QwtSplineC1.

Definition at line 521 of file qwt_spline_local.cpp.

14.153.4.6 type() `QwtSplineLocal::Type QwtSplineLocal::type ( ) const`

Returns

Spline type, specifying the type of interpolation

Definition at line 468 of file qwt_spline_local.cpp.

14.154 QwtSplineParametrization Class Reference

Curve parametrization used for a spline interpolation.

`#include <qwt_spline_parametrization.h>`

Public Types

- enum Type {
  ParameterX , ParameterY , ParameterUniform , ParameterChordal ,
  ParameterCentripetal , ParameterManhattan }

  Parametrization type.

Public Member Functions

- QwtSplineParametrization (int type)
- virtual ~QwtSplineParametrization ()

  Destructor.
- int type () const
- virtual double valueIncrement (const QPointF &, const QPointF &) const

  Calculate the parameter value increment for 2 points.

Static Public Member Functions

- static double valueIncrementX (const QPointF &, const QPointF &)

  Calculate the ParameterX value increment for 2 points.
- static double valueIncrementY (const QPointF &, const QPointF &)

  Calculate the ParameterY value increment for 2 points.
- static double valueIncrementUniform (const QPointF &, const QPointF &)

  Calculate the ParameterUniform value increment.
- static double valueIncrementChordal (const QPointF &, const QPointF &)

  Calculate the ParameterChordal value increment for 2 points.
- static double valueIncrementCentripetal (const QPointF &, const QPointF &)

  Calculate the ParameterCentripetal value increment for 2 points.
- static double valueIncrementManhattan (const QPointF &, const QPointF &)

  Calculate the ParameterManhattan value increment for 2 points.

14.154.1 Detailed Description

Curve parametrization used for a spline interpolation.

Parametrization is the process of finding a parameter value for each curve point - usually related to some physical quantity (distance, time ...).

Often accumulating the curve length is the intended way of parametrization, but as the interpolated curve is not known in advance an approximation needs to be used.

The values are calculated by cumulating increments, that are provided by QwtSplineParametrization. As the curve parameters need to be montonically increasing, each increment need to be positive.

- t[0] = 0;
- t[i] = t[i-1] + valueIncrement(point[i-1], p[i]);

QwtSplineParametrization provides the most common used type of parametrizations and offers an interface to inject custom implementations.

Note

The most relevant types of parametrization are trying to provide an approximation of the curve length.

See also

QwtSpline::setParametrization()

Definition at line 44 of file qwt_spline_parametrization.h.

14.154.2 Member Enumeration Documentation

14.154.2.1 Type enum QwtSplineParametrization::Type

Parametrization type.

Enumerator

ParameterX	No parametrization: t[i] = x[i] **See also** valueIncrementX()
ParameterY	No parametrization: t[i] = y[i] **See also** valueIncrementY()
ParameterUniform	Uniform parametrization: t[i] = i; A very fast parametrization, with good results, when the geometry of the control points is somehow "equidistant". F.e. when recording the position of a body, that is moving with constant speed every n seconds. **See also** valueIncrementUniform()
ParameterChordal	Parametrization using the chordal length between two control points The chordal length is the most commonly used approximation for the curve length. **See also** valueIncrementChordal()
ParameterCentripetal	Centripetal parametrization Based on the square root of the chordal length. Its name stems from the physical observations regarding the centripetal force, of a body moving along the curve. **See also** valueIncrementCentripetal()
ParameterManhattan	Parametrization using the manhattan length between two control points Approximating the curve length by the manhattan length is faster than the chordal length, but usually gives worse results. **See also** valueIncrementManhattan()

Definition at line 48 of file qwt_spline_parametrization.h.

14.154.3 Constructor & Destructor Documentation

14.154.3.1 QwtSplineParametrization() `QwtSplineParametrization::QwtSplineParametrization ( int type ) [explicit]`

Constructor

Parameters

type	Parametrization type

See also

type()

Definition at line 17 of file qwt_spline_parametrization.cpp.

14.154.4 Member Function Documentation

14.154.4.1 type() `int QwtSplineParametrization::type ( ) const`

Returns

Parametrization type

Definition at line 72 of file qwt_spline_parametrization.cpp.

14.154.4.2 valueIncrement() `double QwtSplineParametrization::valueIncrement (`

```
const QPointF & point1,
const QPointF & point2 ) const  [virtual]
```

Calculate the parameter value increment for 2 points.

Parameters

point1	First point
point2	Second point

Returns

Value increment

Definition at line 35 of file qwt_spline_parametrization.cpp.

14.154.4.3 valueIncrementCentripetal() `double QwtSplineParametrization::valueIncrementCentripetal (`

```
const QPointF & point1,
const QPointF & point2 )  [inline], [static]
```

Calculate the ParameterCentripetal value increment for 2 points.

Parameters

point1	First point
point2	Second point

Returns

The square root of a chordal increment

Definition at line 196 of file qwt_spline_parametrization.h.

14.154.4.4 valueIncrementChordal()

```
double QwtSplineParametrization::valueIncrementChordal (
    const QPointF & point1,
    const QPointF & point2 )  [inline], [static]
```

Calculate the ParameterChordal value increment for 2 points.

Parameters

point1	First point
point2	Second point

Returns

qSqrt(dx ∗ dx + dy ∗ dy);

Definition at line 179 of file qwt_spline_parametrization.h.

14.154.4.5 valueIncrementManhattan()

```
double QwtSplineParametrization::valueIncrementManhattan (
    const QPointF & point1,
    const QPointF & point2 )  [inline], [static]
```

Calculate the ParameterManhattan value increment for 2 points.

Parameters

point1	First point
point2	Second point

Returns

| point2.x() - point1.x() | + | point2.y() - point1.y() |

Definition at line 210 of file qwt_spline_parametrization.h.

14.154.4.6 valueIncrementUniform()

```
double QwtSplineParametrization::valueIncrementUniform (
    const QPointF & point1,
    const QPointF & point2 )  [inline], [static]
```

Calculate the ParameterUniform value increment.

Parameters

point1	First point
point2	Second point

Returns

1.0

Definition at line 162 of file qwt_spline_parametrization.h.

14.154.4.7 valueIncrementX()

```
double QwtSplineParametrization::valueIncrementX (
    const QPointF & point1,
    const QPointF & point2 )  [inline], [static]
```

Calculate the ParameterX value increment for 2 points.

Parameters

point1	First point
point2	Second point

Returns

point2.x() - point1.x();

Definition at line 134 of file qwt_spline_parametrization.h.

14.154.4.8 valueIncrementY()

```
double QwtSplineParametrization::valueIncrementY (
    const QPointF & point1,
    const QPointF & point2 )  [inline], [static]
```

Calculate the ParameterY value increment for 2 points.

Parameters

point1	First point
point2	Second point

Returns

point2.y() - point1.y();

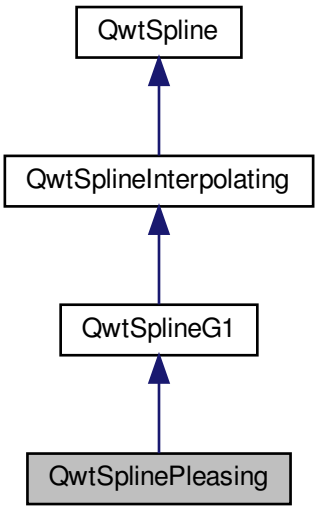
Definition at line 148 of file qwt_spline_parametrization.h.

14.155 QwtSplinePleasing Class Reference

A spline with G1 continuity.

`#include <qwt_spline_pleasing.h>`

Inheritance diagram for QwtSplinePleasing:



Public Member Functions

- QwtSplinePleasing ()
  Constructor.
- virtual ~QwtSplinePleasing ()
  Destructor.
- virtual uint locality () const override
- virtual QPainterPath painterPath (const QPolygonF &) const override
  Interpolate a curve with Bezier curves.
- virtual QVector< QLineF > bezierControlLines (const QPolygonF &) const override
  Interpolate a curve with Bezier curves.

Additional Inherited Members

14.155.1 Detailed Description

A spline with G1 continuity.

QwtSplinePleasing is some sort of cardinal spline, with non C1 continuous extra rules for narrow angles. It has a locality of 2.

Note

The algorithm is the one offered by a popular office package.

Definition at line 23 of file qwt_spline_pleasing.h.

14.155.2 Constructor & Destructor Documentation

14.155.2.1 QwtSplinePleasing() `QwtSplinePleasing::QwtSplinePleasing ( )`

Constructor.

The default setting is a non closing spline with uniform parametrization. (QwtSplineParametrization::ParameterUniform).

See also

QwtSpline::setParametrization(), QwtSpline::setBoundaryType()

Definition at line 265 of file qwt_spline_pleasing.cpp.

14.155.3 Member Function Documentation

14.155.3.1 bezierControlLines() `QVector< QLineF > QwtSplinePleasing::bezierControlLines ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns the 2 control points of each curve as QLineF.

Parameters

points	Control points

Returns

Control points of the interpolating Bezier curves

Implements QwtSplineInterpolating.

Definition at line 327 of file qwt_spline_pleasing.cpp.

14.155.3.2 locality() `uint QwtSplinePleasing::locality ( ) const [override], [virtual]`

Returns

2

Reimplemented from QwtSpline.

Definition at line 276 of file qwt_spline_pleasing.cpp.

14.155.3.3 painterPath() `QPainterPath QwtSplinePleasing::painterPath ( const QPolygonF & points ) const [override], [virtual]`

Interpolate a curve with Bezier curves.

Interpolates a polygon piecewise with cubic Bezier curves and returns them as QPainterPath.

Parameters

points	Control points

Returns

QPainterPath Painter path, that can be rendered by QPainter

Reimplemented from QwtSplineInterpolating.

Definition at line 290 of file qwt_spline_pleasing.cpp.

14.156 QwtSplinePolynomial Class Reference

A cubic polynomial without constant term.

`#include <qwt_spline_polynomial.h>`

Public Member Functions

- QwtSplinePolynomial (double c3=0.0, double c2=0.0, double c1=0.0)

  Constructor.
- bool operator== (const QwtSplinePolynomial &) const
- bool operator!= (const QwtSplinePolynomial &) const
- double valueAt (double x) const
- double slopeAt (double x) const
- double curvatureAt (double x) const

Static Public Member Functions

- static QwtSplinePolynomial fromSlopes (const QPointF &p1, double m1, const QPointF &p2, double m2)
- static QwtSplinePolynomial fromSlopes (double x, double y, double m1, double m2)
- static QwtSplinePolynomial fromCurvatures (const QPointF &p1, double cv1, const QPointF &p2, double cv2)
- static QwtSplinePolynomial fromCurvatures (double dx, double dy, double cv1, double cv2)

Public Attributes

- double c3

  coefficient of the cubic summand
- double c2

  coefficient of the quadratic summand
- double c1

  coefficient of the linear summand

14.156.1 Detailed Description

A cubic polynomial without constant term.

QwtSplinePolynomial is a 3rd degree polynomial of the form: y = c3 $*$ x^3 + c2 $*$ x^2 + c1 $*$ x;

QwtSplinePolynomial is usually used in combination with polygon interpolation, where it is not necessary to store a constant term (c0), as the translation is known from the corresponding polygon points.

See also

QwtSplineC1

Definition at line 30 of file qwt_spline_polynomial.h.

14.156.2 Constructor & Destructor Documentation

14.156.2.1 QwtSplinePolynomial()

```
QwtSplinePolynomial::QwtSplinePolynomial (
        double a3 = 0.0,
        double a2 = 0.0,
        double a1 = 0.0 )  [inline]
```

Constructor.

Parameters

a3	Coefficient of the cubic summand
a2	Coefficient of the quadratic summand
a1	Coefficient of the linear summand

Definition at line 77 of file qwt_spline_polynomial.h.

14.156.3 Member Function Documentation

14.156.3.1 curvatureAt() `double QwtSplinePolynomial::curvatureAt ( double x ) const [inline]`

Calculate the value of the second derivate of a polynomial for a given x

Parameters

x	Parameter

Returns

Curvature at x

Definition at line 130 of file qwt_spline_polynomial.h.

14.156.3.2 fromCurvatures() [1/2] `QwtSplinePolynomial QwtSplinePolynomial::fromCurvatures ( const QPointF & p1, double cv1, const QPointF & p2, double cv2 ) [inline], [static]`

Find the coefficients for the polynomial including 2 points with specific values for the 2nd derivates at these points.

Parameters

p1	First point
cv1	Value of the second derivate at p1
p2	Second point
cv2	Value of the second derivate at p2

Returns

Coefficients of the polynomials

Note

The missing constant term of the polynomial is p1.y()

Definition at line 185 of file qwt_spline_polynomial.h.

14.156.3.3 fromCurvatures() [2/2]

```
QwtSplinePolynomial QwtSplinePolynomial::fromCurvatures (
    double dx,
    double dy,
    double cv1,
    double cv2 )  [inline], [static]
```

Find the coefficients for the polynomial from the offset between 2 points and specific values for the 2nd derivates at these points.

Parameters

dx	X-offset
dy	Y-offset
cv1	Value of the second derivate at p1
cv2	Value of the second derivate at p2

Returns

Coefficients of the polynomials

Definition at line 202 of file qwt_spline_polynomial.h.

14.156.3.4 fromSlopes() [1/2]

```
QwtSplinePolynomial QwtSplinePolynomial::fromSlopes (
    const QPointF & p1,
    double m1,
    const QPointF & p2,
    double m2 )  [inline], [static]
```

Find the coefficients for the polynomial including 2 points with specific values for the 1st derivates at these points.

Parameters

p1	First point
m1	Value of the first derivate at p1
p2	Second point
m2	Value of the first derivate at p2

Returns

Coefficients of the polynomials

Note

The missing constant term of the polynomial is p1.y()

Definition at line 147 of file qwt_spline_polynomial.h.

14.156.3.5 fromSlopes() [2/2]

```
QwtSplinePolynomial QwtSplinePolynomial::fromSlopes (
    double dx,
    double dy,
    double m1,
    double m2 )  [inline], [static]
```

Find the coefficients for the polynomial from the offset between 2 points and specific values for the 1st derivates at these points.

Parameters

dx	X-offset
dy	Y-offset
m1	Value of the first derivate at p1
m2	Value of the first derivate at p2

Returns

Coefficients of the polynomials

Definition at line 164 of file qwt_spline_polynomial.h.

14.156.3.6 operator"!=()

```
bool QwtSplinePolynomial::operator!= (
    const QwtSplinePolynomial & other ) const  [inline]
```

Parameters

other	Other polynomial

Returns

true, when the polynomials have different coefficients

Definition at line 97 of file qwt_spline_polynomial.h.

14.156.3.7 operator==()

```
bool QwtSplinePolynomial::operator== (
    const QwtSplinePolynomial & other ) const  [inline]
```

Parameters

other	Other polynomial

Returns

true, when both polynomials have the same coefficients

Definition at line 88 of file qwt_spline_polynomial.h.

14.156.3.8 slopeAt() `double QwtSplinePolynomial::slopeAt ( double x ) const [inline]`

Calculate the value of the first derivate of a polynomial for a given x

Parameters

x	Parameter

Returns

Slope at x

Definition at line 119 of file qwt_spline_polynomial.h.

14.156.3.9 valueAt() `double QwtSplinePolynomial::valueAt ( double x ) const [inline]`

Calculate the value of a polynomial for a given x

Parameters

x	Parameter

Returns

Value at x

Definition at line 108 of file qwt_spline_polynomial.h.

14.157 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 ,
  Path , Pixmap , Graphic , SvgDocument ,
  UserStyle = 1000 }
- enum CachePolicy { NoCache , Cache , AutoCache }

Public Member Functions

- QwtSymbol (Style=NoSymbol)
- QwtSymbol (Style, const QBrush &, const QPen &, const QSize &)

  Constructor.
- QwtSymbol (const QPainterPath &, const QBrush &, const QPen &)

  Constructor.
- virtual ~QwtSymbol ()

  Destructor.
- void setCachePolicy (CachePolicy)
- CachePolicy cachePolicy () const
- void setSize (const QSize &)
- void setSize (int width, int height=-1)

  Specify the symbol's size.
- const QSize & size () const
- void setPinPoint (const QPointF &pos, bool enable=true)

  Set and enable a pin point.
- QPointF pinPoint () const
- void setPinPointEnabled (bool)
- bool isPinPointEnabled () const
- virtual void setColor (const QColor &)

  Set the color of the symbol.
- void setBrush (const QBrush &)

  Assign a brush.
- const QBrush & brush () const
- void setPen (const QColor &, qreal width=0.0, Qt::PenStyle=Qt::SolidLine)
- void setPen (const QPen &)
- const QPen & pen () const
- void setStyle (Style)
- Style style () const
- void setPath (const QPainterPath &)

  Set a painter path as symbol.
- const QPainterPath & path () const
- void setPixmap (const QPixmap &)
- const QPixmap & pixmap () const
- void setGraphic (const QwtGraphic &)
- const QwtGraphic & graphic () const
- void setSvgDocument (const QByteArray &)
- void drawSymbol (QPainter *, const QRectF &) const

  Draw the symbol into a rectangle.
- void drawSymbol (QPainter *, const QPointF &) const

  Draw the symbol at a specified position.

- void drawSymbols (QPainter *, const QPolygonF &) const

  Draw symbols at the specified points.
- void drawSymbols (QPainter *, const QPointF *, int numPoints) const
- virtual QRect boundingRect () const
- void invalidateCache ()

Protected Member Functions

- virtual void renderSymbols (QPainter *, const QPointF *, int numPoints) const

14.157.1 Detailed Description

A class for drawing symbols.

Definition at line 31 of file qwt_symbol.h.

14.157.2 Member Enumeration Documentation

14.157.2.1 CachePolicy `enum QwtSymbol::CachePolicy`

Depending on the render engine and the complexity of the symbol shape it might be faster to render the symbol to a pixmap and to paint this pixmap.

F.e. the raster paint engine is a pure software renderer where in cache mode a draw operation usually ends in raster operation with the the backing store, that are usually faster, than the algorithms for rendering polygons. But the opposite can be expected for graphic pipelines that can make use of hardware acceleration.

The default setting is AutoCache

See also

setCachePolicy(), cachePolicy()

Note

The policy has no effect, when the symbol is painted to a vector graphics format (PDF, SVG).

Warning

Since Qt 4.8 raster is the default backend on X11

Enumerator

NoCache	Don't use a pixmap cache.
Cache	Always use a pixmap cache.
AutoCache	Use a cache when one of the following conditions is true: • The symbol is rendered with the software renderer (QPaintEngine::Raster)

Definition at line 150 of file qwt_symbol.h.

14.157.2.2 Style enum QwtSymbol::Style

Symbol Style

See also

setStyle(), style()

Enumerator

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.
Path	The symbol is represented by a painter path, where the origin (0, 0) of the path coordinate system is mapped to the position of the symbol. **See also** setPath(), path()
Pixmap	The symbol is represented by a pixmap. The pixmap is centered or aligned to its pin point. **See also** setPinPoint()
Graphic	The symbol is represented by a graphic. The graphic is centered or aligned to its pin point. **See also** setPinPoint()
SvgDocument	The symbol is represented by a SVG graphic. The graphic is centered or aligned to its pin point. **See also** setPinPoint()
UserStyle	Styles >= QwtSymbol::UserSymbol are reserved for derived classes of QwtSymbol that overload drawSymbols() with additional application specific symbol types.

Definition at line 38 of file qwt_symbol.h.

14.157.3 Constructor & Destructor Documentation

14.157.3.1 QwtSymbol() [1/3] QwtSymbol::QwtSymbol (Style *style* = *NoSymbol*) [explicit]

Default Constructor

Parameters

style	Symbol Style

The symbol is constructed with gray interior, black outline with zero width, no size and style 'NoSymbol'.

Definition at line 843 of file qwt_symbol.cpp.

14.157.3.2 QwtSymbol() [2/3] QwtSymbol::QwtSymbol (QwtSymbol::Style *style*, const QBrush & *brush*, const QPen & *pen*, const QSize & *size*)

Constructor.

Parameters

style	Symbol Style
brush	brush to fill the interior
pen	outline pen
size	size

See also

setStyle(), setBrush(), setPen(), setSize()

Definition at line 858 of file qwt_symbol.cpp.

14.157.3.3 QwtSymbol() [3/3] QwtSymbol::QwtSymbol (const QPainterPath & *path*, const QBrush & *brush*, const QPen & *pen*)

Constructor.

The symbol gets initialized by a painter path. The style is set to QwtSymbol::Path, the size is set to empty (the path is displayed unscaled).

Parameters

path	painter path
brush	brush to fill the interior
pen	outline pen

See also

setPath(), setBrush(), setPen(), setSize()

Definition at line 878 of file qwt_symbol.cpp.

14.157.4 Member Function Documentation

14.157.4.1 boundingRect() `QRect QwtSymbol::boundingRect ( ) const [virtual]`

Calculate the bounding rectangle for a symbol at position (0,0).

Returns

Bounding rectangle

Definition at line 1637 of file qwt_symbol.cpp.

14.157.4.2 brush() `const QBrush & QwtSymbol::brush ( ) const`

Returns

Brush

See also

setBrush()

Definition at line 1123 of file qwt_symbol.cpp.

14.157.4.3 cachePolicy() `QwtSymbol::CachePolicy QwtSymbol::cachePolicy ( ) const`

Returns

Cache policy

See also

CachePolicy, setCachePolicy()

Definition at line 913 of file qwt_symbol.cpp.

14.157.4.4 drawSymbol() `[1/2]`

```
void QwtSymbol::drawSymbol (
    QPainter * painter,
    const QPointF & pos ) const [inline]
```

Draw the symbol at a specified position.

Parameters

painter	Painter
pos	Position of the symbol in screen coordinates

Definition at line 238 of file qwt_symbol.h.

14.157.4.5 drawSymbol() [2/2]

```
void QwtSymbol::drawSymbol (
    QPainter * painter,
    const QRectF & rect ) const
```

Draw the symbol into a rectangle.

The symbol is painted centered and scaled into the target rectangle. It is always painted uncached and the pin point is ignored.

This method is primarily intended for drawing a symbol to the legend.

Parameters

painter	Painter
rect	Target rectangle for the symbol

Definition at line 1434 of file qwt_symbol.cpp.

14.157.4.6 drawSymbols() [1/2]

```
void QwtSymbol::drawSymbols (
    QPainter * painter,
    const QPointF * points,
    int numPoints ) const
```

Render an array of symbols

Painting several symbols is more effective than drawing symbols one by one, as a couple of layout calculations and setting of pen/brush can be done once for the complete array.

Parameters

painter	Painter
points	Array of points
numPoints	Number of points

Definition at line 1307 of file qwt_symbol.cpp.

14.157.4.7 drawSymbols() [2/2] void QwtSymbol::drawSymbols (QPainter * *painter*, const QPolygonF & *points*) const [inline]

Draw symbols at the specified points.

Parameters

painter	Painter
points	Positions of the symbols in screen coordinates

Definition at line 251 of file qwt_symbol.h.

14.157.4.8 graphic() const QwtGraphic & QwtSymbol::graphic ( ) const

Returns

Assigned graphic

See also

setGraphic()

Definition at line 1027 of file qwt_symbol.cpp.

14.157.4.9 invalidateCache() void QwtSymbol::invalidateCache ( )

Invalidate the cached symbol pixmap

The symbol invalidates its cache, whenever an attribute is changed that has an effect ob how to display a symbol. In case of derived classes with individual styles (>= QwtSymbol::UserStyle) it might be necessary to call invalidateCache() for attributes that are relevant for this style.

See also

CachePolicy, setCachePolicy(), drawSymbols()

Definition at line 1770 of file qwt_symbol.cpp.

14.157.4.10 isPinPointEnabled() `bool QwtSymbol::isPinPointEnabled ( ) const`

Returns

True, when the pin point translation is enabled

See also

setPinPoint(), setPinPointEnabled()

Definition at line 1291 of file qwt_symbol.cpp.

14.157.4.11 path() `const QPainterPath & QwtSymbol::path ( ) const`

Returns

Painter path for displaying the symbol

See also

setPath()

Definition at line 977 of file qwt_symbol.cpp.

14.157.4.12 pen() `const QPen & QwtSymbol::pen ( ) const`

Returns

Pen

See also

setPen(), brush()

Definition at line 1171 of file qwt_symbol.cpp.

14.157.4.13 pinPoint() `QPointF QwtSymbol::pinPoint ( ) const`

Returns

Pin point

See also

setPinPoint(), setPinPointEnabled()

Definition at line 1267 of file qwt_symbol.cpp.

14.157.4.14 pixmap() `const QPixmap & QwtSymbol::pixmap ( ) const`

Returns

Assigned pixmap

See also

setPixmap()

Definition at line 1002 of file qwt_symbol.cpp.

14.157.4.15 renderSymbols() `void QwtSymbol::renderSymbols ( QPainter * painter, const QPointF * points, int numPoints ) const [protected], [virtual]`

Render the symbol to series of points

Parameters

painter	Qt painter
points	Positions of the symbols
numPoints	Number of points

Definition at line 1513 of file qwt_symbol.cpp.

14.157.4.16 setBrush() `void QwtSymbol::setBrush ( const QBrush & brush )`

Assign a brush.

The brush is used to draw the interior of the symbol.

Parameters

brush	Brush

See also

brush()

Definition at line 1107 of file qwt_symbol.cpp.

14.157.4.17 setCachePolicy() void QwtSymbol::setCachePolicy (QwtSymbol::CachePolicy *policy*)

Change the cache policy

The default policy is AutoCache

Parameters

policy	Cache policy

See also

CachePolicy, cachePolicy()

Definition at line 899 of file qwt_symbol.cpp.

14.157.4.18 setColor() void QwtSymbol::setColor (const QColor & *color*) [virtual]

Set the color of the symbol.

Change the color of the brush for symbol types with a filled area. For all other symbol types the color will be assigned to the pen.

Parameters

color	Color

See also

setBrush(), setPen(), brush(), pen()

Definition at line 1186 of file qwt_symbol.cpp.

14.157.4.19 setGraphic() void QwtSymbol::setGraphic (const QwtGraphic & *graphic*)

Set a graphic as symbol

Parameters

graphic	Graphic

See also

graphic(), setPixmap()

Note

the style() is set to QwtSymbol::Graphic

brush() and pen() have no effect

Definition at line 1017 of file qwt_symbol.cpp.

14.157.4.20 setPath() `void QwtSymbol::setPath ( const QPainterPath & path )`

Set a painter path as symbol.

The symbol is represented by a painter path, where the origin (0, 0) of the path coordinate system is mapped to the position of the symbol.

When the symbol has valid size the painter path gets scaled to fit into the size. Otherwise the symbol size depends on the bounding rectangle of the path.

Example

The following code defines a symbol drawing an arrow:

```
#include <qwt_symbol.h>
     QwtSymbol *symbol = new QwtSymbol();
     QPen pen( Qt::black, 2 );
     pen.setJoinStyle( Qt::MiterJoin );
     symbol->setPen( pen );
     symbol->setBrush( Qt::red );
     QPainterPath path;
     path.moveTo( 0, 8 );
     path.lineTo( 0, 5 );
     path.lineTo( -3, 5 );
     path.lineTo( 0, 0 );
     path.lineTo( 3, 5 );
     path.lineTo( 0, 5 );
     QTransform transform;
     transform.rotate( -30.0 );
     path = transform.map( path );
     symbol->setPath( path );
     symbol->setPinPoint( QPointF( 0.0, 0.0 ) );
     setSize( 10, 14 );
```

Parameters

path	Painter path

Note

The style is implicitly set to QwtSymbol::Path.

See also

path(), setSize()

Definition at line 966 of file qwt_symbol.cpp.

14.157.4.21 setPen() [1/2] void QwtSymbol::setPen (const QColor & color, qreal width = 0.0, Qt::PenStyle style = Qt::SolidLine)

Build and assign a pen

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

Parameters

color	Pen color
width	Pen width
style	Pen style

See also

pen(), brush()

Definition at line 1141 of file qwt_symbol.cpp.

14.157.4.22 setPen() [2/2] void QwtSymbol::setPen (const QPen & pen)

Assign a pen

The pen is used to draw the symbol's outline.

Parameters

pen	Pen

See also

pen(), setBrush()

Definition at line 1155 of file qwt_symbol.cpp.

14.157.4.23 setPinPoint() void QwtSymbol::setPinPoint (const QPointF & pos, bool enable = true)

Set and enable a pin point.

The position of a complex symbol is not always aligned to its center (f.e an arrow, where the peak points to a position). The pin point defines the position inside of a Pixmap, Graphic, SvgDocument or PainterPath symbol where the represented point has to be aligned to.

Parameters

pos	Position
enable	En/Disable the pin point alignment

See also

pinPoint(), setPinPointEnabled()

Definition at line 1249 of file qwt_symbol.cpp.

14.157.4.24 setPinPointEnabled() `void QwtSymbol::setPinPointEnabled ( bool on )`

En/Disable the pin point alignment

Parameters

on	Enabled, when on is true

See also

setPinPoint(), isPinPointEnabled()

Definition at line 1278 of file qwt_symbol.cpp.

14.157.4.25 setPixmap() `void QwtSymbol::setPixmap ( const QPixmap & pixmap )`

Set a pixmap as symbol

Parameters

pixmap	Pixmap

See also

pixmap(), setGraphic()

Note

the style() is set to QwtSymbol::Pixmap

brush() and pen() have no effect

Definition at line 992 of file qwt_symbol.cpp.

14.157.4.26 setSize() [1/2] void QwtSymbol::setSize (const QSize & *size*)

Set the symbol's size

Parameters

size	Size

See also

size()

Definition at line 1081 of file qwt_symbol.cpp.

14.157.4.27 setSize() [2/2] 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

width	Width
height	Height (defaults to -1)

See also

size()

Definition at line 1067 of file qwt_symbol.cpp.

14.157.4.28 setStyle() void QwtSymbol::setStyle (QwtSymbol::Style *style*)

Specify the symbol style

Parameters

style	Style

See also

style()

Definition at line 1782 of file qwt_symbol.cpp.

14.157.4.29 setSvgDocument() `void QwtSymbol::setSvgDocument ( const QByteArray & svgDocument )`

Set a SVG icon as symbol

Parameters

svgDocument	SVG icon

See also

setGraphic(), setPixmap()

Note

the style() is set to QwtSymbol::SvgDocument

brush() and pen() have no effect

Definition at line 1044 of file qwt_symbol.cpp.

14.157.4.30 size() `const QSize & QwtSymbol::size ( ) const`

Returns

Size

See also

setSize()

Definition at line 1094 of file qwt_symbol.cpp.

14.157.4.31 style() `QwtSymbol::Style QwtSymbol::style ( ) const`

Returns

Current symbol style

See also

setStyle()

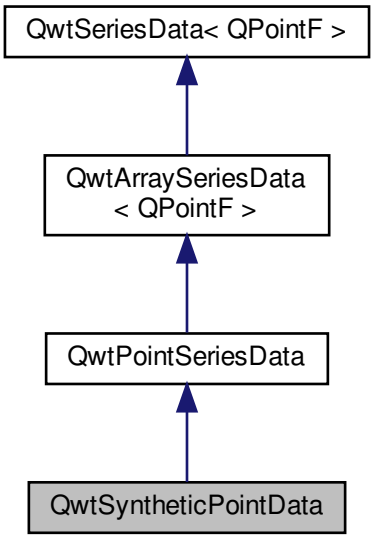
Definition at line 1795 of file qwt_symbol.cpp.

14.158 QwtSyntheticPointData Class Reference

Synthetic point data.

`#include <qwt_point_data.h>`

Inheritance diagram for QwtSyntheticPointData:



Public Member Functions

- QwtSyntheticPointData (size_t size, const QwtInterval &=QwtInterval())
- void setSize (size_t size)
- virtual size_t size () const override
- void setInterval (const QwtInterval &)
- QwtInterval interval () const
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.
- virtual QPointF sample (size_t index) const override
- virtual double y (double x) const =0
- virtual double x (uint index) const
- virtual void setRectOfInterest (const QRectF &) override
- QRectF rectOfInterest () const

Additional Inherited Members

14.158.1 Detailed Description

Synthetic point data.

QwtSyntheticPointData provides a fixed number of points for an interval. The points are calculated in equidistant steps in x-direction.

If the interval is invalid, the points are calculated for the "rectangle of interest", what normally is the displayed area on the plot canvas. In this mode you get different levels of detail, when zooming in/out.

Example

The following example shows how to implement a sinus curve.

```
#include <cmath>
#include <qwt_series_data.h>
#include <qwt_plot_curve.h>
#include <qwt_plot.h>
#include <qapplication.h>
  class SinusData: public QwtSyntheticPointData
  {
  public:
   SinusData():
       QwtSyntheticPointData( 100 )
   {
   }
   virtual double y( double x ) const
   {
       return qSin( x );
   }
  };
  int main(int argc, char **argv)
  {
   QApplication a( argc, argv );
   QwtPlot plot;
   plot.setAxisScale( QwtAxis::XBottom, 0.0, 10.0 );
   plot.setAxisScale( QwtAxis::YLeft, -1.0, 1.0 );
   QwtPlotCurve *curve = new QwtPlotCurve( "y = sin(x)" );
   curve->setData( new SinusData() );
   curve->attach( &plot );
   plot.show();
   return a.exec();
  }
```

Definition at line 157 of file qwt_point_data.h.

14.158.2 Constructor & Destructor Documentation

14.158.2.1 QwtSyntheticPointData() `QwtSyntheticPointData::QwtSyntheticPointData (`

```
size_t size,
const QwtInterval & interval = QwtInterval() )
```

Constructor

Parameters

size	Number of points
interval	Bounding interval for the points

See also

setInterval(), setSize()

Definition at line 20 of file qwt_point_data.cpp.

14.158.3 Member Function Documentation

14.158.3.1 boundingRect() `QRectF QwtSyntheticPointData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

This implementation iterates over all points, what could often be implemented much faster using the characteristics of the series. When there are many points it is recommended to overload and reimplement this method using the characteristics of the series (if possible).

Returns

Bounding rectangle

Reimplemented from QwtPointSeriesData.

Definition at line 105 of file qwt_point_data.cpp.

14.158.3.2 interval() `QwtInterval QwtSyntheticPointData::interval ( ) const`

Returns

Bounding interval

See also

setInterval(), size()

Definition at line 62 of file qwt_point_data.cpp.

14.158.3.3 rectOfInterest() `QRectF QwtSyntheticPointData::rectOfInterest ( ) const`

Returns

"rectangle of interest"

See also

setRectOfInterest()

Definition at line 89 of file qwt_point_data.cpp.

14.158.3.4 sample() `QPointF QwtSyntheticPointData::sample ( size_t index ) const [override], [virtual]`

Calculate the point from an index

Parameters

index	Index

Returns

QPointF(x(index), y(x(index)));

Warning

For invalid indices (index < 0 || index >= size()) (0, 0) is returned.

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 125 of file qwt_point_data.cpp.

14.158.3.5 setInterval() `void QwtSyntheticPointData::setInterval ( const QwtInterval & interval )`

Set the bounding interval

Parameters

interval	Interval

See also

interval(), setSize()

Definition at line 53 of file qwt_point_data.cpp.

14.158.3.6 setRectOfInterest() `void QwtSyntheticPointData::setRectOfInterest ( const QRectF & rect ) [override], [virtual]`

Set a the "rectangle of interest"

QwtPlotSeriesItem defines the current area of the plot canvas as "rect of interest" (QwtPlotSeriesItem::updateScaleDiv()).

If interval().isValid() == false the x values are calculated in the interval rect.left() -> rect.right().

See also

rectOfInterest()

Reimplemented from QwtSeriesData< QPointF >.

Definition at line 78 of file qwt_point_data.cpp.

14.158.3.7 setSize() void QwtSyntheticPointData::setSize (size_t *size*)

Change the number of points

Parameters

size	Number of points

See also

size(), setInterval()

Definition at line 33 of file qwt_point_data.cpp.

14.158.3.8 size() size_t QwtSyntheticPointData::size ( ) const [override], [virtual]

Returns

Number of points

See also

setSize(), interval()

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 42 of file qwt_point_data.cpp.

14.158.3.9 x() double QwtSyntheticPointData::x (uint *index*) const [virtual]

Calculate a x-value from an index

x values are calculated by dividing an interval into equidistant steps. If !interval().isValid() the interval is calculated from the "rectangle of interest".

Parameters

index	Index of the requested point

Returns

Calculated x coordinate

See also

interval(), rectOfInterest(), y()

Definition at line 148 of file qwt_point_data.cpp.

14.158.3.10 y() `virtual double QwtSyntheticPointData::y ( double x ) const [pure virtual]`

Calculate a y value for a x value

Parameters

x	x value

Returns

Corresponding y value

14.159 QwtSystemClock Class Reference

QwtSystemClock provides high resolution clock time functions.

`#include <qwt_system_clock.h>`

Public Member Functions

- bool isNull () const
- void start ()

  Start the elapsed timer.
- double restart ()
- double elapsed () const

14.159.1 Detailed Description

QwtSystemClock provides high resolution clock time functions.

Precision and time intervals are multiples of milliseconds (ms).

(QwtSystemClock is deprecated as QElapsedTimer offers the same precision)

Definition at line 24 of file qwt_system_clock.h.

14.159.2 Member Function Documentation

14.159.2.1 elapsed() `double QwtSystemClock::elapsed ( ) const`

Returns

elapsed time in multiples of milliseconds

Definition at line 36 of file qwt_system_clock.cpp.

14.159.2.2 isNull() `bool QwtSystemClock::isNull ( ) const`

Returns

true, if the elapsed timer is valid

Definition at line 14 of file qwt_system_clock.cpp.

14.159.2.3 restart() `double QwtSystemClock::restart ( )`

Restart the elapsed timer

Returns

elapsed time in multiples of milliseconds

Definition at line 29 of file qwt_system_clock.cpp.

14.160 QwtText Class Reference

A class representing a text.

`#include <qwt_text.h>`

Public Types

- enum TextFormat { AutoText = 0 , PlainText , RichText , MathMLText , TeXText , OtherFormat = 100 }
  Text format.
- enum PaintAttribute { PaintUsingTextFont = 0x01 , PaintUsingTextColor = 0x02 , PaintBackground = 0x04 }
  Paint Attributes.
- enum LayoutAttribute { MinimumLayout = 0x01 }
  Layout Attributes The layout attributes affects some aspects of the layout of the text.
- typedef QFlags< PaintAttribute > PaintAttributes
- typedef QFlags< LayoutAttribute > LayoutAttributes

Public Member Functions

- QwtText ()
- QwtText (const QString &, TextFormat textFormat=AutoText)
- QwtText (const QwtText &)

  Copy constructor.
- ~QwtText ()

  Destructor.
- QwtText & operator= (const QwtText &)

  Assignment operator.
- bool operator== (const QwtText &) const

  Relational operator.
- bool operator!= (const QwtText &) const

  Relational operator.
- void setText (const QString &, QwtText::TextFormat textFormat=AutoText)
- QString text () const
- bool isNull () const
- bool isEmpty () const
- void setFont (const QFont &)
- QFont font () const

  Return the font.
- QFont usedFont (const QFont &) const
- void setRenderFlags (int)

  Change the render flags.
- int renderFlags () const
- void setColor (const QColor &)
- QColor color () const

  Return the pen color, used for painting the text.
- QColor usedColor (const QColor &) const
- void setBorderRadius (double)
- double borderRadius () const
- void setBorderPen (const QPen &)
- QPen borderPen () const
- void setBackgroundBrush (const QBrush &)
- QBrush backgroundBrush () const
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- void setLayoutAttribute (LayoutAttribute, bool on=true)
- bool testLayoutAttribute (LayoutAttribute) const
- double heightForWidth (double width) const
- double heightForWidth (double width, const QFont &) const
- QSizeF textSize () const
- QSizeF textSize (const QFont &) const
- void draw (QPainter *painter, const QRectF &rect) const

Static Public Member Functions

- static const QwtTextEngine * textEngine (const QString &text, QwtText::TextFormat=AutoText)
- static const QwtTextEngine * textEngine (QwtText::TextFormat)

  Find the text engine for a text format.
- static void setTextEngine (QwtText::TextFormat, QwtTextEngine *)

14.160.1 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 special QwtTextEngine for this format.
- Background
  A text might have a background, defined by a QPen and QBrush to improve its visibility. The corners of the background might be rounded.
- Font
  A text might have an individual font.
- Color
  A text might have an individual color.
- Render Flags
  Flags from Qt::AlignmentFlag and Qt::TextFlag used like in QPainter::drawText().

See also

QwtTextEngine, QwtTextLabel

Definition at line 51 of file qwt_text.h.

14.160.2 Member Typedef Documentation

14.160.2.1 LayoutAttributes `typedef QFlags<LayoutAttribute > QwtText::LayoutAttributes`

An ORed combination of LayoutAttribute values.

Definition at line 143 of file qwt_text.h.

14.160.2.2 PaintAttributes `typedef QFlags<PaintAttribute > QwtText::PaintAttributes`

An ORed combination of PaintAttribute values.

Definition at line 126 of file qwt_text.h.

14.160.3 Member Enumeration Documentation

14.160.3.1 LayoutAttribute `enum QwtText::LayoutAttribute`

Layout Attributes The layout attributes affects some aspects of the layout of the text.

Enumerator

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.

Definition at line 132 of file qwt_text.h.

14.160.3.2 PaintAttribute `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.

Enumerator

PaintUsingTextFont	The text has an individual font.
PaintUsingTextColor	The text has an individual color.
PaintBackground	The text has an individual background.

Definition at line 114 of file qwt_text.h.

14.160.3.3 TextFormat `enum QwtText::TextFormat`

Text format.

The text format defines the QwtTextEngine, that is used to render the text.

See also

QwtTextEngine, setTextEngine()

Enumerator

AutoText	The text format is determined using QwtTextEngine::mightRender() for all available text engines in increasing order > PlainText. If none of the text engines can render the text is rendered like QwtText::PlainText.
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. In earlier versions of Qwt such an engine was included - since Qwt 6.2 it can be found here: https://github.com/uwerat/qwt-mml-dev To enable MathML support the following code needs to be added to the application: `QwtText::setTextEngine( QwtText::MathMLText, new QwtMathMLTextEngine() );`
TeXText	Use a TeX (http://en.wikipedia.org/wiki/TeX) render engine to display the text (not implemented yet).
OtherFormat	The number of text formats can be extended using setTextEngine. Formats >= QwtText::OtherFormat are not used by Qwt.

Definition at line 64 of file qwt_text.h.

14.160.4 Constructor & Destructor Documentation

14.160.4.1 QwtText() [1/2] QwtText::QwtText ( )

Constructor

Definition at line 201 of file qwt_text.cpp.

14.160.4.2 QwtText() [2/2] QwtText::QwtText (const QString & *text*, QwtText::TextFormat *textFormat* = *AutoText*)

Constructor

Parameters

text	Text content
textFormat	Text format

Definition at line 215 of file qwt_text.cpp.

14.160.5 Member Function Documentation

14.160.5.1 backgroundBrush() QBrush QwtText::backgroundBrush ( ) const

Returns

Background brush

See also

setBackgroundBrush(), borderPen()

Definition at line 451 of file qwt_text.cpp.

14.160.5.2 borderPen() `QPen QwtText::borderPen ( ) const`

Returns

Background pen

See also

setBorderPen(), backgroundBrush()

Definition at line 430 of file qwt_text.cpp.

14.160.5.3 borderRadius() `double QwtText::borderRadius ( ) const`

Returns

Radius for the corners of the border frame

See also

setBorderRadius(), borderPen(), backgroundBrush()

Definition at line 409 of file qwt_text.cpp.

14.160.5.4 draw()

```
void QwtText::draw (
    QPainter * painter,
    const QRectF & rect ) const
```

Draw a text into a rectangle

Parameters

painter	Painter
rect	Rectangle

Definition at line 615 of file qwt_text.cpp.

14.160.5.5 heightForWidth() [1/2]

```
double QwtText::heightForWidth (
    double width ) const
```

Find the height for a given width

Parameters

width	Width

Returns

Calculated height

Definition at line 522 of file qwt_text.cpp.

14.160.5.6 heightForWidth() [2/2]

```
double QwtText::heightForWidth (
    double width,
    const QFont & defaultFont ) const
```

Find the height for a given width

Parameters

defaultFont	Font, used for the calculation if the text has no font
width	Width

Returns

Calculated height

Definition at line 535 of file qwt_text.cpp.

14.160.5.7 isEmpty()

```
bool QwtText::isEmpty ( ) const
```

Returns

text().isEmpty()

Definition at line 739 of file qwt_text.cpp.

14.160.5.8 isNull()

```
bool QwtText::isNull ( ) const
```

Returns

text().isNull()

Definition at line 733 of file qwt_text.cpp.

14.160.5.9 renderFlags() `int QwtText::renderFlags ( ) const`

Returns

Render flags

See also

setRenderFlags()

Definition at line 317 of file qwt_text.cpp.

14.160.5.10 setBackgroundBrush() `void QwtText::setBackgroundBrush ( const QBrush & brush )`

Set the background brush

Parameters

brush	Background brush

See also

backgroundBrush(), setBorderPen()

Definition at line 441 of file qwt_text.cpp.

14.160.5.11 setBorderPen() `void QwtText::setBorderPen ( const QPen & pen )`

Set the background pen

Parameters

pen	Background pen

See also

borderPen(), setBackgroundBrush()

Definition at line 420 of file qwt_text.cpp.

14.160.5.12 setBorderRadius() `void QwtText::setBorderRadius ( double radius )`

Set the radius for the corners of the border frame

Parameters

radius	Radius of a rounded corner

See also

borderRadius(), setBorderPen(), setBackgroundBrush()

Definition at line 400 of file qwt_text.cpp.

14.160.5.13 setColor()

```
void QwtText::setColor (
    const QColor & color )
```

Set the pen color used for drawing the text.

Parameters

color	Color

Note

Setting the color might have no effect, when the text contains control sequences for setting colors.

Definition at line 365 of file qwt_text.cpp.

14.160.5.14 setFont()

```
void QwtText::setFont (
    const QFont & font )
```

Set the font.

Parameters

font	Font

Note

Setting the font might have no effect, when the text contains control sequences for setting fonts.

Definition at line 329 of file qwt_text.cpp.

14.160.5.15 setLayoutAttribute()

```
void QwtText::setLayoutAttribute (
    LayoutAttribute attribute,
    bool on = true )
```

Change a layout attribute

Parameters

attribute	Layout attribute
on	On/Off

See also

testLayoutAttribute()

Definition at line 494 of file qwt_text.cpp.

14.160.5.16 setPaintAttribute() `void QwtText::setPaintAttribute ( PaintAttribute attribute, bool on = true )`

Change a paint attribute

Parameters

attribute	Paint attribute
on	On/Off

Note

Used by setFont(), setColor(), setBorderPen() and setBackgroundBrush()

See also

testPaintAttribute()

Definition at line 466 of file qwt_text.cpp.

14.160.5.17 setRenderFlags() `void QwtText::setRenderFlags ( int renderFlags )`

Change the render flags.

The default setting is Qt::AlignCenter

Parameters

renderFlags	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.

Definition at line 304 of file qwt_text.cpp.

14.160.5.18 setText()

```
void QwtText::setText (
    const QString & text,
    QwtText::TextFormat textFormat = AutoText )
```

Assign a new text content

Parameters

text	Text content
textFormat	Text format

See also

text()

Definition at line 277 of file qwt_text.cpp.

14.160.5.19 setTextEngine()

```
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.

For QwtText::PlainText it is not allowed to assign a engine == NULL.

Parameters

format	Text format
engine	Text engine

Warning

Using QwtText::AutoText does nothing.

Definition at line 713 of file qwt_text.cpp.

14.160.5.20 testLayoutAttribute() `bool QwtText::testLayoutAttribute ( LayoutAttribute attribute ) const`

Test a layout attribute

Parameters

attribute	Layout attribute

Returns

true, if attribute is enabled

See also

setLayoutAttribute()

Definition at line 510 of file qwt_text.cpp.

14.160.5.21 testPaintAttribute() `bool QwtText::testPaintAttribute ( PaintAttribute attribute ) const`

Test a paint attribute

Parameters

attribute	Paint attribute

Returns

true, if attribute is enabled

See also

setPaintAttribute()

Definition at line 482 of file qwt_text.cpp.

14.160.5.22 text() `QString QwtText::text ( ) const`

Returns

Text as QString.

See also

setText()

Definition at line 289 of file qwt_text.cpp.

14.160.5.23 textEngine() [1/2]

```
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

text	Text, needed in case of AutoText
format	Text format

Returns

Corresponding text engine

Definition at line 694 of file qwt_text.cpp.

14.160.5.24 textEngine() [2/2]

```
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.

Parameters

format	Text format

Returns

The text engine, or NULL if no engine is available.

Definition at line 727 of file qwt_text.cpp.

14.160.5.25 textSize() [1/2] QSizeF QwtText::textSize ( ) const

Returns the size, that is needed to render text

Returns

Calculated size

Definition at line 570 of file qwt_text.cpp.

14.160.5.26 textSize() [2/2] QSizeF QwtText::textSize (const QFont & *defaultFont*) const

Returns the size, that is needed to render text

Parameters

defaultFont	Font of the text

Returns

Calculated size

Definition at line 581 of file qwt_text.cpp.

14.160.5.27 usedColor() QColor QwtText::usedColor (const QColor & *defaultColor*) const

Return the color of the text, if it has one. Otherwise return defaultColor.

Parameters

defaultColor	Default color

Returns

Color used for drawing the text

See also

setColor(), color(), PaintAttributes

Definition at line 386 of file qwt_text.cpp.

14.160.5.28 usedFont() `QFont QwtText::usedFont ( const QFont & defaultFont ) const`

Return the font of the text, if it has one. Otherwise return defaultFont.

Parameters

defaultFont	Default font

Returns

Font used for drawing the text

See also

setFont(), font(), PaintAttributes

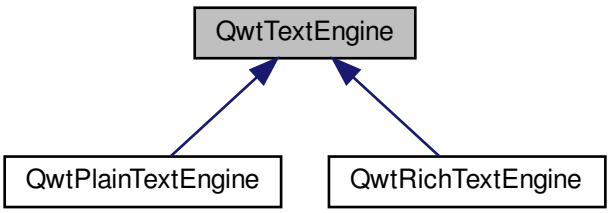
Definition at line 350 of file qwt_text.cpp.

14.161 QwtTextEngine Class Reference

Abstract base class for rendering text strings.

`#include <qwt_text_engine.h>`

Inheritance diagram for QwtTextEngine:



Public Member Functions

- virtual ~QwtTextEngine ()
  Destructor.
- virtual double heightForWidth (const QFont &font, int flags, const QString &text, double width) const =0
- virtual QSizeF textSize (const QFont &font, int flags, const QString &text) const =0
- virtual bool mightRender (const QString &text) const =0
- virtual void textMargins (const QFont &font, const QString &text, double &left, double &right, double &top, double &bottom) const =0
- virtual void draw (QPainter *painter, const QRectF &rect, int flags, const QString &text) const =0

Protected Member Functions

- QwtTextEngine ()
  Constructor.

14.161.1 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.

See also

QwtText::setTextEngine()

Definition at line 30 of file qwt_text_engine.h.

14.161.2 Member Function Documentation

14.161.2.1 draw()

```
virtual void QwtTextEngine::draw (
    QPainter * painter,
    const QRectF & rect,
    int flags,
    const QString & text ) const  [pure virtual]
```

Draw the text in a clipping rectangle

Parameters

painter	Painter
rect	Clipping rectangle
flags	Bitwise OR of the flags like in for QPainter::drawText()
text	Text to be rendered

Implemented in QwtRichTextEngine, and QwtPlainTextEngine.

14.161.2.2 heightForWidth()

```
virtual double QwtTextEngine::heightForWidth (
    const QFont & font,
    int flags,
    const QString & text,
    double width ) const  [pure virtual]
```

Find the height for a given width

Parameters

font	Font of the text
flags	Bitwise OR of the flags used like in QPainter::drawText
text	Text to be rendered
width	Width

Returns

Calculated height

Implemented in QwtRichTextEngine, and QwtPlainTextEngine.

14.161.2.3 mightRender()

```
virtual bool QwtTextEngine::mightRender (
    const QString & text ) const  [pure virtual]
```

Test if a string can be rendered by this text engine

Parameters

text	Text to be tested

Returns

true, if it can be rendered

Implemented in QwtRichTextEngine, and QwtPlainTextEngine.

14.161.2.4 textMargins()

```
virtual void QwtTextEngine::textMargins (
    const QFont & font,
    const QString & text,
    double & left,
    double & right,
    double & top,
    double & 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

font	Font of the text
text	Text to be rendered
left	Return value for the left margin
right	Return value for the right margin
top	Return value for the top margin
bottom	Return value for the bottom margin

Implemented in QwtRichTextEngine, and QwtPlainTextEngine.

14.161.2.5 textSize()

```
virtual QSizeF QwtTextEngine::textSize (
    const QFont & font,
    int flags,
    const QString & text ) const  [pure virtual]
```

Returns the size, that is needed to render text

Parameters

font	Font of the text
flags	Bitwise OR of the flags like in for QPainter::drawText
text	Text to be rendered

Returns

Calculated size

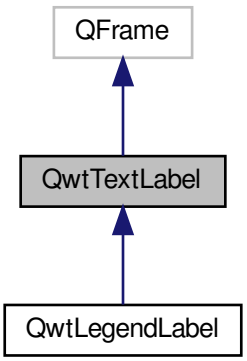
Implemented in QwtRichTextEngine, and QwtPlainTextEngine.

14.162 QwtTextLabel Class Reference

A Widget which displays a QwtText.

```
#include <qwt_text_label.h>
```

Inheritance diagram for QwtTextLabel:



Public Slots

- void setText (const QString &, QwtText::TextFormat textFormat=QwtText::AutoText)
- virtual void setText (const QwtText &)
- void clear ()

  Clear the text and all QwtText attributes.

Public Member Functions

- QwtTextLabel (QWidget *parent=NULL)
- QwtTextLabel (const QwtText &, QWidget *parent=NULL)
- virtual ~QwtTextLabel ()

  Destructor.
- void setPlainText (const QString &)
- QString plainText () const
- const QwtText & text () const

  Return the text.
- int indent () const

  Return label's text indent in pixels.
- void setIndent (int)
- int margin () const

  Return label's text margin in pixels.
- void setMargin (int)
- virtual QSize sizeHint () const override

  Return a size hint.
- virtual QSize minimumSizeHint () const override

  Return a minimum size hint.
- virtual int heightForWidth (int) const override
- QRect textRect () const
- virtual void drawText (QPainter *, const QRectF &)

  Redraw the text.

Protected Member Functions

- virtual void paintEvent (QPaintEvent *) override
- virtual void drawContents (QPainter *)

  Redraw the text and focus indicator.

14.162.1 Detailed Description

A Widget which displays a QwtText.

Definition at line 26 of file qwt_text_label.h.

14.162.2 Constructor & Destructor Documentation

14.162.2.1 QwtTextLabel() [1/2] QwtTextLabel::QwtTextLabel (QWidget * *parent* = *NULL*) [explicit]

Constructs an empty label.

Parameters

parent	Parent widget

Definition at line 39 of file qwt_text_label.cpp.

14.162.2.2 QwtTextLabel() [2/2] QwtTextLabel::QwtTextLabel (const QwtText & *text*, QWidget * *parent* = *NULL*) [explicit]

Constructs a label that displays the text, text

Parameters

parent	Parent widget
text	Text

Definition at line 50 of file qwt_text_label.cpp.

14.162.3 Member Function Documentation

14.162.3.1 heightForWidth() int QwtTextLabel::heightForWidth (int *width*) const [override], [virtual]

Parameters

width	Width

Returns

Preferred height for this widget, given the width.

Definition at line 209 of file qwt_text_label.cpp.

14.162.3.2 paintEvent() void QwtTextLabel::paintEvent (QPaintEvent * *event*) [override], [protected], [virtual]

Qt paint event

Parameters

event	Paint event

Reimplemented in QwtLegendLabel.

Definition at line 236 of file qwt_text_label.cpp.

14.162.3.3 plainText() `QString QwtTextLabel::plainText ( ) const`

Interface for the designer plugin

Returns

Text as plain text

See also

setPlainText(), text()

Definition at line 84 of file qwt_text_label.cpp.

14.162.3.4 setIndent() `void QwtTextLabel::setIndent ( int indent )`

Set label's text indent in pixels

Parameters

indent	Indentation in pixels

Definition at line 142 of file qwt_text_label.cpp.

14.162.3.5 setMargin() `void QwtTextLabel::setMargin ( int margin )`

Set label's margin in pixels

Parameters

margin	Margin in pixels

Definition at line 163 of file qwt_text_label.cpp.

14.162.3.6 setPlainText() `void QwtTextLabel::setPlainText ( const QString & text )`

Interface for the designer plugin - does the same as setText()

See also

plainText()

Definition at line 73 of file qwt_text_label.cpp.

14.162.3.7 setText [1/2]

```
void QwtTextLabel::setText (
    const QString & text,
    QwtText::TextFormat textFormat = QwtText::AutoText )  [slot]
```

Change the label's text, keeping all other QwtText attributes

Parameters

text	New text
textFormat	Format of text

See also

QwtText

Definition at line 96 of file qwt_text_label.cpp.

14.162.3.8 setText [2/2]

```
void QwtTextLabel::setText (
    const QwtText & text )  [virtual], [slot]
```

Change the label's text

Parameters

text	New text

Reimplemented in QwtLegendLabel.

Definition at line 109 of file qwt_text_label.cpp.

14.162.3.9 textRect()

```
QRect QwtTextLabel::textRect ( ) const
```

Calculate geometry for the text in widget coordinates

Returns

Geometry for the text

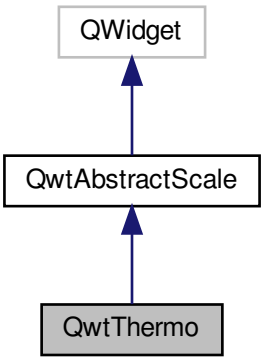
Definition at line 288 of file qwt_text_label.cpp.

14.163 QwtThermo Class Reference

The Thermometer Widget.

`#include <qwt_thermo.h>`

Inheritance diagram for QwtThermo:



Public Types

- enum ScalePosition { NoScale , LeadingScale , TrailingScale }
- enum OriginMode { OriginMinimum , OriginMaximum , OriginCustom }

Public Slots

- virtual void setValue (double)

Public Member Functions

- QwtThermo (QWidget *parent=NULL)
- virtual ~QwtThermo ()

  Destructor.
- void setOrientation (Qt::Orientation)

  Set the orientation.
- Qt::Orientation orientation () const
- void setScalePosition (ScalePosition)

  Change the position of the scale.
- ScalePosition scalePosition () const
- void setSpacing (int)

  Change the spacing between pipe and scale.
- int spacing () const
- void setBorderWidth (int)
- int borderWidth () const

- void setOriginMode (OriginMode)

  Change how the origin is determined.
- OriginMode originMode () const
- void setOrigin (double)

  Specifies the custom origin.
- double origin () const
- void setFillBrush (const QBrush &)

  Change the brush of the liquid.
- QBrush fillBrush () const
- void setAlarmBrush (const QBrush &)

  Specify the liquid brush above the alarm threshold.
- QBrush alarmBrush () const
- void setAlarmLevel (double)
- double alarmLevel () const
- void setAlarmEnabled (bool)

  Enable or disable the alarm threshold.
- bool alarmEnabled () const
- void setColorMap (QwtColorMap ∗)

  Assign a color map for the fill color.
- QwtColorMap ∗ colorMap ()
- const QwtColorMap ∗ colorMap () const
- void setPipeWidth (int)
- int pipeWidth () const
- void setRangeFlags (QwtInterval::BorderFlags)

  Exclude/Include min/max values.
- QwtInterval::BorderFlags rangeFlags () const
- double value () const

  Return the value.
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override
- void setScaleDraw (QwtScaleDraw ∗)

  Set a scale draw.
- const QwtScaleDraw ∗ scaleDraw () const

Protected Member Functions

- virtual void drawLiquid (QPainter ∗, const QRect &) const
- virtual void scaleChange () override

  Notify a scale change.
- virtual void paintEvent (QPaintEvent ∗) override
- virtual void resizeEvent (QResizeEvent ∗) override
- virtual void changeEvent (QEvent ∗) override
- QwtScaleDraw ∗ scaleDraw ()
- QRect pipeRect () const
- QRect fillRect (const QRect &) const

  Calculate the filled rectangle of the pipe.
- QRect alarmRect (const QRect &) const

  Calculate the alarm rectangle of the pipe.

14.163.1 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.

The fill colors might be calculated from an optional color map If no color map has been assigned QwtThermo uses the following colors/brushes from the widget palette:

- QPalette::Base Background of the pipe
- QPalette::ButtonText Fill brush below the alarm level
- QPalette::Highlight Fill brush for the values above the alarm level
- QPalette::WindowText For the axis of the scale
- QPalette::Text For the labels of the scale

Definition at line 46 of file qwt_thermo.h.

14.163.2 Member Enumeration Documentation

14.163.2.1 OriginMode `enum QwtThermo::OriginMode`

Origin mode. This property specifies where the beginning of the liquid is placed.

See also

setOriginMode(), setOrigin()

Enumerator

OriginMinimum	The origin is the minimum of the scale.
OriginMaximum	The origin is the maximum of the scale.
OriginCustom	The origin is specified using the origin() property.

Definition at line 91 of file qwt_thermo.h.

14.163.2.2 ScalePosition `enum QwtThermo::ScalePosition`

Position of the scale

See also

setScalePosition(), setOrientation()

Enumerator

NoScale	The slider has no scale.
LeadingScale	The scale is right of a vertical or below of a horizontal slider.
TrailingScale	The scale is left of a vertical or above of a horizontal slider.

Definition at line 73 of file qwt_thermo.h.

14.163.3 Constructor & Destructor Documentation

14.163.3.1 QwtThermo() `QwtThermo::QwtThermo ( QWidget * parent = NULL ) [explicit]`

Constructor

Parameters

parent	Parent widget

Definition at line 121 of file qwt_thermo.cpp.

14.163.4 Member Function Documentation

14.163.4.1 alarmBrush() `QBrush QwtThermo::alarmBrush ( ) const`

Returns

Liquid brush (QPalette::Highlight) above the alarm threshold.

See also

setAlarmBrush(), QWidget::palette()

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 761 of file qwt_thermo.cpp.

14.163.4.2 alarmEnabled() `bool QwtThermo::alarmEnabled ( ) const`

Returns

True, when the alarm threshold is enabled.

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 837 of file qwt_thermo.cpp.

14.163.4.3 alarmLevel() `double QwtThermo::alarmLevel ( ) const`

Returns

Alarm threshold.

See also

setAlarmLevel()

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 789 of file qwt_thermo.cpp.

14.163.4.4 alarmRect() `QRect QwtThermo::alarmRect ( const QRect & fillRect ) const [protected]`

Calculate the alarm rectangle of the pipe.

Parameters

fillRect	Filled rectangle in the pipe

Returns

Rectangle to be filled with the alarm brush

See also

pipeRect(), fillRect(), alarmLevel(), alarmBrush()

Definition at line 944 of file qwt_thermo.cpp.

14.163.4.5 borderWidth() `int QwtThermo::borderWidth ( ) const`

Returns

Border width of the thermometer pipe.

See also

setBorderWidth()

Definition at line 671 of file qwt_thermo.cpp.

14.163.4.6 changeEvent() `void QwtThermo::changeEvent ( QEvent * event ) [override], [protected], [virtual]`

Qt change event handler

Parameters

event	Event

Reimplemented from QwtAbstractScale.

Definition at line 277 of file qwt_thermo.cpp.

14.163.4.7 colorMap() [1/2] `QwtColorMap * QwtThermo::colorMap ( )`

Returns

Color map for the fill color

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 697 of file qwt_thermo.cpp.

14.163.4.8 colorMap() [2/2] const QwtColorMap * QwtThermo::colorMap ( ) const

Returns

Color map for the fill color

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 707 of file qwt_thermo.cpp.

14.163.4.9 drawLiquid() void QwtThermo::drawLiquid (QPainter * *painter*, const QRect & *pipeRect*) const [protected], [virtual]

Redraw the liquid in thermometer pipe.

Parameters

painter	Painter
pipeRect	Bounding rectangle of the pipe without borders

Definition at line 546 of file qwt_thermo.cpp.

14.163.4.10 fillBrush() QBrush QwtThermo::fillBrush ( ) const

Returns

Liquid (QPalette::ButtonText) brush.

See also

setFillBrush(), QWidget::palette()

Definition at line 731 of file qwt_thermo.cpp.

14.163.4.11 fillRect() QRect QwtThermo::fillRect (const QRect & *pipeRect*) const [protected]

Calculate the filled rectangle of the pipe.

Parameters

pipeRect	Rectangle of the pipe

Returns

Rectangle to be filled (fill and alarm brush)

See also

pipeRect(), alarmRect()

Definition at line 897 of file qwt_thermo.cpp.

14.163.4.12 minimumSizeHint() `QSize QwtThermo::minimumSizeHint ( ) const [override], [virtual]`

Returns

Minimum size hint

Warning

The return value depends on the font and the scale.

See also

sizeHint()

Definition at line 856 of file qwt_thermo.cpp.

14.163.4.13 orientation() `Qt::Orientation QwtThermo::orientation ( ) const`

Returns

Orientation

See also

setOrientation()

Definition at line 455 of file qwt_thermo.cpp.

14.163.4.14 origin() `double QwtThermo::origin ( ) const`

Returns

Origin of the thermo, when OriginCustom is enabled

See also

setOrigin(), setOriginMode(), originMode()

Definition at line 504 of file qwt_thermo.cpp.

14.163.4.15 originMode() `QwtThermo::OriginMode QwtThermo::originMode ( ) const`

Returns

Mode, how the origin is determined.

See also

setOriginMode(), serOrigin(), origin()

Definition at line 477 of file qwt_thermo.cpp.

14.163.4.16 paintEvent() `void QwtThermo::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Paint event handler

Parameters

event	Paint event

Definition at line 235 of file qwt_thermo.cpp.

14.163.4.17 pipeRect() `QRect QwtThermo::pipeRect ( ) const [protected]`

Returns

Bounding rectangle of the pipe (without borders) in widget coordinates

Definition at line 385 of file qwt_thermo.cpp.

14.163.4.18 pipeWidth() `int QwtThermo::pipeWidth ( ) const`

Returns

Width of the pipe.

See also

setPipeWidth()

Definition at line 813 of file qwt_thermo.cpp.

14.163.4.19 rangeFlags() `QwtInterval::BorderFlags QwtThermo::rangeFlags ( ) const`

Returns

Range flags

See also

setRangeFlags()

Definition at line 170 of file qwt_thermo.cpp.

14.163.4.20 resizeEvent() `void QwtThermo::resizeEvent ( QResizeEvent * event ) [override], [protected], [virtual]`

Resize event handler

Parameters

event	Resize event

Definition at line 267 of file qwt_thermo.cpp.

14.163.4.21 scaleDraw() [1/2] `QwtScaleDraw * QwtThermo::scaleDraw ( ) [protected]`

Returns

the scale draw of the thermo

See also

setScaleDraw()

Definition at line 226 of file qwt_thermo.cpp.

14.163.4.22 scaleDraw() [2/2] const QwtScaleDraw * QwtThermo::scaleDraw ( ) const

Returns

the scale draw of the thermo

See also

setScaleDraw()

Definition at line 217 of file qwt_thermo.cpp.

14.163.4.23 scalePosition() QwtThermo::ScalePosition QwtThermo::scalePosition ( ) const

Returns

Scale position.

See also

setScalePosition()

Definition at line 530 of file qwt_thermo.cpp.

14.163.4.24 setAlarmBrush() void QwtThermo::setAlarmBrush (const QBrush & *brush*)

Specify the liquid brush above the alarm threshold.

Changes the QPalette::Highlight brush of the palette.

Parameters

brush	New brush.

See also

alarmBrush(), QWidget::setPalette()

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 747 of file qwt_thermo.cpp.

14.163.4.25 setAlarmEnabled() `void QwtThermo::setAlarmEnabled ( bool on )`

Enable or disable the alarm threshold.

Parameters

on	true (disabled) or false (enabled)

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 825 of file qwt_thermo.cpp.

14.163.4.26 setAlarmLevel() `void QwtThermo::setAlarmLevel ( double level )`

Specify the alarm threshold.

Parameters

level	Alarm threshold

See also

alarmLevel()

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 775 of file qwt_thermo.cpp.

14.163.4.27 setBorderWidth() void QwtThermo::setBorderWidth (int *width*)

Set the border width of the pipe.

Parameters

width	Border width

See also

borderWidth()

Definition at line 655 of file qwt_thermo.cpp.

14.163.4.28 setColorMap() void QwtThermo::setColorMap (QwtColorMap * *colorMap*)

Assign a color map for the fill color.

Parameters

colorMap	Color map

Warning

The alarm threshold has no effect, when a color map has been assigned

Definition at line 683 of file qwt_thermo.cpp.

14.163.4.29 setFillBrush() void QwtThermo::setFillBrush (const QBrush & *brush*)

Change the brush of the liquid.

Changes the QPalette::ButtonText brush of the palette.

Parameters

brush	New brush.

See also

fillBrush(), QWidget::setPalette()

Definition at line 720 of file qwt_thermo.cpp.

14.163.4.30 setOrientation() void QwtThermo::setOrientation (Qt::Orientation *orientation*)

Set the orientation.

Parameters

orientation	Allowed values are Qt::Horizontal and Qt::Vertical.

See also

orientation(), scalePosition()

Definition at line 432 of file qwt_thermo.cpp.

14.163.4.31 setOrigin() void QwtThermo::setOrigin (double *origin*)

Specifies the custom origin.

If originMode is set to OriginCustom this property controls where the liquid starts.

Parameters

origin	New origin level

See also

setOriginMode(), originMode(), origin()

Definition at line 491 of file qwt_thermo.cpp.

14.163.4.32 setOriginMode() void QwtThermo::setOriginMode (OriginMode *m*)

Change how the origin is determined.

See also

originMode(), serOrigin(), origin()

Definition at line 464 of file qwt_thermo.cpp.

14.163.4.33 setPipeWidth() void QwtThermo::setPipeWidth (int *width*)

Change the width of the pipe.

Parameters

width	Width of the pipe

See also

pipeWidth()

Definition at line 800 of file qwt_thermo.cpp.

14.163.4.34 setRangeFlags() `void QwtThermo::setRangeFlags ( QwtInterval::BorderFlags flags )`

Exclude/Include min/max values.

According to the flags minValue() and maxValue() are included/excluded from the pipe. In case of an excluded value the corresponding tick is painted 1 pixel off of the pipeRect().

F.e. when a minimum of 0.0 has to be displayed as an empty pipe the minValue() needs to be excluded.

Parameters

flags	Range flags

See also

rangeFlags()

Definition at line 157 of file qwt_thermo.cpp.

14.163.4.35 setScaleDraw() `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

scaleDraw	ScaleDraw object, that has to be created with new and will be deleted in ~QwtThermo() or the next call of setScaleDraw().

Definition at line 207 of file qwt_thermo.cpp.

14.163.4.36 setScalePosition() void QwtThermo::setScalePosition (ScalePosition *scalePosition*)

Change the position of the scale.

Parameters

scalePosition	Position of the scale.

See also

ScalePosition, scalePosition()

Definition at line 515 of file qwt_thermo.cpp.

14.163.4.37 setSpacing() void QwtThermo::setSpacing (int *spacing*)

Change the spacing between pipe and scale.

A spacing of 0 means, that the backbone of the scale is below the pipe.

The default setting is 3 pixels.

Parameters

spacing	Number of pixels

See also

spacing();

Definition at line 629 of file qwt_thermo.cpp.

14.163.4.38 setValue void QwtThermo::setValue (double *value*) [virtual], [slot]

Set the current value.

Parameters

value	New Value

See also

value()

Definition at line 181 of file qwt_thermo.cpp.

14.163.4.39 sizeHint() `QSize QwtThermo::sizeHint ( ) const [override], [virtual]`

Returns

the minimum size hint

See also

minimumSizeHint()

Definition at line 846 of file qwt_thermo.cpp.

14.163.4.40 spacing() `int QwtThermo::spacing ( ) const`

Returns

Number of pixels between pipe and scale

See also

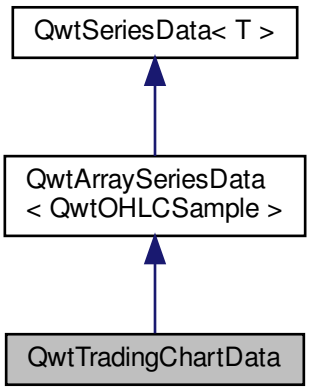
setSpacing()

Definition at line 645 of file qwt_thermo.cpp.

14.164 QwtTradingChartData Class Reference

`#include <qwt_series_data.h>`

Inheritance diagram for QwtTradingChartData:



Public Member Functions

- QwtTradingChartData (const QVector< QwtOHLCSample > &=QVector< QwtOHLCSample >())
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.

Additional Inherited Members

14.164.1 Detailed Description

Interface for iterating over an array of OHLC samples

Definition at line 261 of file qwt_series_data.h.

14.164.2 Constructor & Destructor Documentation

14.164.2.1 QwtTradingChartData() `QwtTradingChartData::QwtTradingChartData ( const QVector< QwtOHLCSample > & samples = QVector< QwtOHLCSample >( ) )`

Constructor

Parameters

samples	Samples

Definition at line 378 of file qwt_series_data.cpp.

14.164.3 Member Function Documentation

14.164.3.1 boundingRect() `QRectF QwtTradingChartData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

Implements QwtSeriesData< T >.

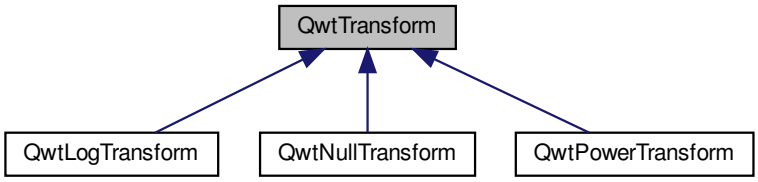
Definition at line 391 of file qwt_series_data.cpp.

14.165 QwtTransform Class Reference

A transformation between coordinate systems.

`#include <qwt_transform.h>`

Inheritance diagram for QwtTransform:



Public Member Functions

- QwtTransform ()
  Constructor.
- virtual ~QwtTransform ()
  Destructor.
- virtual double bounded (double value) const
- virtual double transform (double value) const =0
- virtual double invTransform (double value) const =0
- virtual QwtTransform * copy () const =0
  Virtualized copy operation.

14.165.1 Detailed Description

A transformation between coordinate systems.

QwtTransform manipulates values, when being mapped between the scale and the paint device coordinate system.

A transformation consists of 2 methods:

- transform
- invTransform

where one is is the inverse function of the other.

When p1, p2 are the boundaries of the paint device coordinates and s1, s2 the boundaries of the scale, QwtScaleMap uses the following calculations:

- p = p1 + (p2 - p1) * (T(s) - T(s1) / (T(s2) - T(s1));
- s = invT (T(s1) + (T(s2) - T(s1)) * (p - p1) / (p2 - p1));

Definition at line 35 of file qwt_transform.h.

14.165.2 Member Function Documentation

14.165.2.1 bounded() `double QwtTransform::bounded ( double value ) const [virtual]`

Modify value to be a valid value for the transformation. The default implementation does nothing.

Parameters

value	Value to be bounded

Returns

value unmodified

Reimplemented in QwtLogTransform.

Definition at line 33 of file qwt_transform.cpp.

14.165.2.2 invTransform() `virtual double QwtTransform::invTransform ( double value ) const [pure virtual]`

Inverse transformation function

Parameters

value	Value

Returns

Modified value

See also

transform()

Implemented in QwtPowerTransform, QwtLogTransform, and QwtNullTransform.

14.165.2.3 transform() `virtual double QwtTransform::transform ( double value ) const [pure virtual]`

Transformation function

Parameters

value	Value

Returns

Modified value

See also

invTransform()

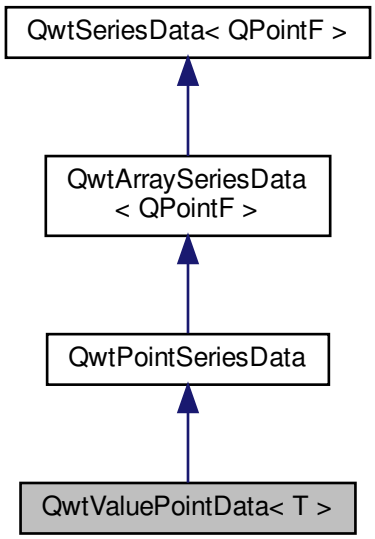
Implemented in QwtPowerTransform, QwtLogTransform, and QwtNullTransform.

14.166 QwtValuePointData< T > Class Template Reference

Interface for iterating over a QVector<T>.

`#include <qwt_point_data.h>`

Inheritance diagram for QwtValuePointData< T >:



Public Member Functions

- QwtValuePointData (const QVector< T > &y)
- QwtValuePointData (const T *y, size_t size)
- virtual size_t size () const override
- virtual QPointF sample (size_t index) const override
- const QVector< T > & yData () const

Additional Inherited Members

14.166.1 Detailed Description

template<typename T>
class QwtValuePointData< T >

Interface for iterating over a QVector<T>.

The memory contains the y coordinates, while the index is interpreted as x coordinate.

Definition at line 67 of file qwt_point_data.h.

14.166.2 Constructor & Destructor Documentation

14.166.2.1 QwtValuePointData() [1/2] template<typename T >

```
QwtValuePointData< T >::QwtValuePointData (
            const QVector< T > & y )
```

Constructor

Parameters

y	Array of y values

See also

QwtPlotCurve::setData(), QwtPlotCurve::setSamples()

Definition at line 266 of file qwt_point_data.h.

14.166.2.2 QwtValuePointData() [2/2] template<typename T >

```
QwtValuePointData< T >::QwtValuePointData (
            const T * y,
            size_t size )
```

Constructor

Parameters

y	Array of y values
size	Size of the x and y arrays

See also

QwtPlotCurve::setData(), QwtPlotCurve::setSamples()

Definition at line 279 of file qwt_point_data.h.

14.166.3 Member Function Documentation

14.166.3.1 sample() template<typename T >

```
QPointF QwtValuePointData< T >::sample (
            size_t index ) const  [override], [virtual]
```

Return the sample at position i

Parameters

index	Index

Returns

Sample at position i

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 299 of file qwt_point_data.h.

14.166.3.2 size() template<typename T >

```
size_t QwtValuePointData< T >::size  [override], [virtual]
```

Returns

Size of the data set

Reimplemented from QwtArraySeriesData< QPointF >.

Definition at line 287 of file qwt_point_data.h.

14.166.3.3 yData() template<typename T >

```
const QVector< T > & QwtValuePointData< T >::yData
```

Returns

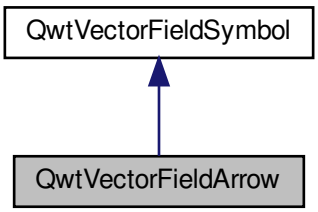
Array of the y-values

Definition at line 306 of file qwt_point_data.h.

14.167 QwtVectorFieldArrow Class Reference

`#include <qwt_vectorfield_symbol.h>`

Inheritance diagram for QwtVectorFieldArrow:



Public Member Functions

- QwtVectorFieldArrow (qreal headWidth=6.0, qreal tailWidth=1.0)
  Constructor.
- virtual ~QwtVectorFieldArrow () override
  Destructor.
- virtual void setLength (qreal length) override
- virtual qreal length () const override
- virtual void paint (QPainter *) const override
  Draw the symbol/arrow.

14.167.1 Detailed Description

Arrow implementation that draws a filled arrow with outline, using a triangular head of constant width.

Definition at line 61 of file qwt_vectorfield_symbol.h.

14.167.2 Constructor & Destructor Documentation

14.167.2.1 QwtVectorFieldArrow() `QwtVectorFieldArrow::QwtVectorFieldArrow (`

```
qreal headWidth = 6.0,
qreal tailWidth = 1.0 )
```

Constructor.

The length is initialized by headWidth + 4

Parameters

headWidth	Width of the triangular head
tailWidth	Width of the arrow tail

See also

setLength()

Definition at line 74 of file qwt_vectorfield_symbol.cpp.

14.167.3 Member Function Documentation

14.167.3.1 length() `qreal QwtVectorFieldArrow::length ( ) const [override], [virtual]`

Returns

length of the symbol/arrow

See also

setLength()

Implements QwtVectorFieldSymbol.

Definition at line 90 of file qwt_vectorfield_symbol.cpp.

14.167.3.2 setLength() `void QwtVectorFieldArrow::setLength ( qreal length ) [override], [virtual]`

Set the length of the symbol/arrow

See also

length()

Implements QwtVectorFieldSymbol.

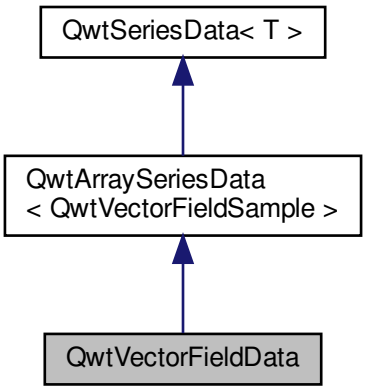
Definition at line 85 of file qwt_vectorfield_symbol.cpp.

14.168 QwtVectorFieldData Class Reference

Interface for iterating over an array of vector field samples.

`#include <qwt_series_data.h>`

Inheritance diagram for QwtVectorFieldData:



Public Member Functions

- QwtVectorFieldData (const QVector< QwtVectorFieldSample > &=QVector< QwtVectorFieldSample >())
- virtual QRectF boundingRect () const override

  Calculate the bounding rectangle.

Additional Inherited Members

14.168.1 Detailed Description

Interface for iterating over an array of vector field samples.

Definition at line 249 of file qwt_series_data.h.

14.168.2 Constructor & Destructor Documentation

14.168.2.1 QwtVectorFieldData() `QwtVectorFieldData::QwtVectorFieldData (const QVector< QwtVectorFieldSample > & samples = QVector< QwtVectorFieldSample >( ) )`

Constructor

Parameters

samples	Samples

Definition at line 327 of file qwt_series_data.cpp.

14.168.3 Member Function Documentation

14.168.3.1 boundingRect() `QRectF QwtVectorFieldData::boundingRect ( ) const [override], [virtual]`

Calculate the bounding rectangle.

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

Implements QwtSeriesData< T >.

Definition at line 341 of file qwt_series_data.cpp.

14.169 QwtVectorFieldSample Class Reference

Sample used in vector fields.

`#include <qwt_samples.h>`

Public Member Functions

- QwtVectorFieldSample (double x=0.0, double y=0.0, double vx=0.0, double vy=0.0)
  Constructor.
- QwtVectorFieldSample (const QPointF &pos, double vx=0.0, double vy=0.0)
  Constructor.
- QPointF pos () const
- bool isNull () const

Public Attributes

- double x
  x coordinate of the position
- double y
  y coordinate of the position
- double vx
  x coordinate of the vector
- double vy
  y coordinate of the vector

14.169.1 Detailed Description

Sample used in vector fields.

A vector field sample is a position and a vector - usually representing some direction and magnitude - attached to this position.

See also

QwtVectorFieldData

Definition at line 243 of file qwt_samples.h.

14.169.2 Constructor & Destructor Documentation

14.169.2.1 QwtVectorFieldSample() [1/2]

```
QwtVectorFieldSample::QwtVectorFieldSample (
    double posX = 0.0,
    double posY = 0.0,
    double vectorX = 0.0,
    double vectorY = 0.0 )  [inline]
```

Constructor.

Parameters

posX	x coordinate of the position
posY	y coordinate of the position
vectorX	x coordinate of the vector
vectorY	y coordinate of the vector

Definition at line 277 of file qwt_samples.h.

14.169.2.2 QwtVectorFieldSample() [2/2]

```
QwtVectorFieldSample::QwtVectorFieldSample (
    const QPointF & pos,
    double vectorX = 0.0,
    double vectorY = 0.0 )  [inline]
```

Constructor.

Parameters

pos	Position
vectorX	x coordinate of the vector
vectorY	y coordinate of the vector

Definition at line 293 of file qwt_samples.h.

14.169.3 Member Function Documentation

14.169.3.1 isNull() `bool QwtVectorFieldSample::isNull ( ) const [inline]`

Returns

true, if vx and vy are 0

Definition at line 309 of file qwt_samples.h.

14.169.3.2 pos() `QPointF QwtVectorFieldSample::pos ( ) const [inline]`

Returns

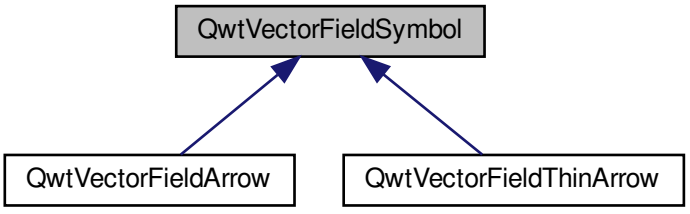
x/y coordinates as QPointF

Definition at line 303 of file qwt_samples.h.

14.170 QwtVectorFieldSymbol Class Reference

`#include <qwt_vectorfield_symbol.h>`

Inheritance diagram for QwtVectorFieldSymbol:



Public Member Functions

- QwtVectorFieldSymbol ()

  Constructor.
- virtual ~QwtVectorFieldSymbol ()

  Destructor.
- virtual void setLength (qreal length)=0
- virtual qreal length () const =0
- virtual void paint (QPainter *) const =0

  Draw the symbol/arrow.

14.170.1 Detailed Description

Defines abstract interface for arrow drawing routines.

Arrow needs to be drawn horizontally with arrow tip at coordinate 0,0. arrowLength() shall return the entire length of the arrow (needed to translate the arrow for tail/centered alignment). setArrowLength() defines arror length in pixels (screen coordinates). It can be implemented to adjust other geometric properties such as the head size and width of the arrow. It is *always* called before paint().

A new arrow implementation can be set with QwtPlotVectorField::setArrowSymbol(), whereby ownership is transferred to the plot field.

Definition at line 32 of file qwt_vectorfield_symbol.h.

14.170.2 Member Function Documentation

14.170.2.1 length() `virtual qreal QwtVectorFieldSymbol::length ( ) const [pure virtual]`

Returns

length of the symbol/arrow

See also

setLength()

Implemented in QwtVectorFieldThinArrow, and QwtVectorFieldArrow.

14.170.2.2 setLength() `virtual void QwtVectorFieldSymbol::setLength ( qreal length ) [pure virtual]`

Set the length of the symbol/arrow

See also

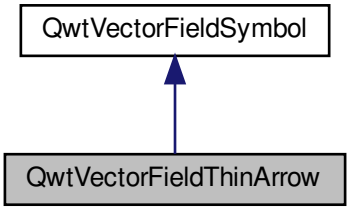
length()

Implemented in QwtVectorFieldThinArrow, and QwtVectorFieldArrow.

14.171 QwtVectorFieldThinArrow Class Reference

`#include <qwt_vectorfield_symbol.h>`

Inheritance diagram for QwtVectorFieldThinArrow:



Public Member Functions

- QwtVectorFieldThinArrow (qreal headWidth=6.0)
  Constructor.
- virtual ~QwtVectorFieldThinArrow () override
  Destructor.
- virtual void setLength (qreal length) override
- virtual qreal length () const override
- virtual void paint (QPainter *) const override
  Draw the symbol/arrow.

14.171.1 Detailed Description

Arrow implementation that only used lines, with optionally a filled arrow or only lines.

Definition at line 81 of file qwt_vectorfield_symbol.h.

14.171.2 Constructor & Destructor Documentation

14.171.2.1 QwtVectorFieldThinArrow() `QwtVectorFieldThinArrow::QwtVectorFieldThinArrow ( qreal headWidth = 6.0 )`

Constructor.

The length is initialized by headWidth + 4

Parameters

headWidth	Width of the triangular head

See also

setLength()

Definition at line 128 of file qwt_vectorfield_symbol.cpp.

14.171.3 Member Function Documentation

14.171.3.1 length() `qreal QwtVectorFieldThinArrow::length ( ) const [override], [virtual]`

Returns

length of the symbol/arrow

See also

setLength()

Implements QwtVectorFieldSymbol.

Definition at line 152 of file qwt_vectorfield_symbol.cpp.

14.171.3.2 setLength() `void QwtVectorFieldThinArrow::setLength ( qreal length ) [override], [virtual]`

Set the length of the symbol/arrow

See also

length()

Implements QwtVectorFieldSymbol.

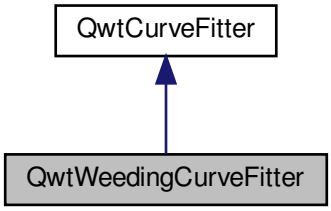
Definition at line 139 of file qwt_vectorfield_symbol.cpp.

14.172 QwtWeedingCurveFitter Class Reference

A curve fitter implementing Douglas and Peucker algorithm.

`#include <qwt_weeding_curve_fitter.h>`

Inheritance diagram for QwtWeedingCurveFitter:



Public Member Functions

- QwtWeedingCurveFitter (double tolerance=1.0)
- virtual ~QwtWeedingCurveFitter ()

  Destructor.
- void setTolerance (double)
- double tolerance () const
- void setChunkSize (uint)
- uint chunkSize () const
- virtual QPolygonF fitCurve (const QPolygonF &) const override
- virtual QPainterPath fitCurvePath (const QPolygonF &) const override

Additional Inherited Members

14.172.1 Detailed Description

A curve fitter implementing Douglas and Peucker algorithm.

The purpose of the Douglas and Peucker algorithm is that given a 'curve' composed of line segments to find a curve not too dissimilar but that has fewer points. The algorithm defines 'too dissimilar' based on the maximum distance (tolerance) between the original curve and the smoothed curve.

The runtime of the algorithm increases non linear (worst case O(n∗n)) and might be very slow for huge polygons. To avoid performance issues it might be useful to split the polygon (setChunkSize()) and to run the algorithm for these smaller parts. The disadvantage of having no interpolation at the borders is for most use cases irrelevant.

The smoothed curve consists of a subset of the points that defined the original curve.

In opposite to QwtSplineCurveFitter the Douglas and Peucker algorithm reduces the number of points. By adjusting the tolerance parameter according to the axis scales QwtSplineCurveFitter can be used to implement different level of details to speed up painting of curves of many points.

Definition at line 38 of file qwt_weeding_curve_fitter.h.

14.172.2 Constructor & Destructor Documentation

14.172.2.1 QwtWeedingCurveFitter() `QwtWeedingCurveFitter::QwtWeedingCurveFitter ( double tolerance = 1.0 ) [explicit]`

Constructor

Parameters

tolerance	Tolerance

See also

setTolerance(), tolerance()

Definition at line 50 of file qwt_weeding_curve_fitter.cpp.

14.172.3 Member Function Documentation

14.172.3.1 chunkSize() `uint QwtWeedingCurveFitter::chunkSize ( ) const`

Returns

Maximum for the number of points passed to a run of the algorithm - or 0, when unlimited

See also

setChunkSize()

Definition at line 114 of file qwt_weeding_curve_fitter.cpp.

14.172.3.2 fitCurve() `QPolygonF QwtWeedingCurveFitter::fitCurve ( const QPolygonF & points ) const [override], [virtual]`

Parameters

points	Series of data points

Returns

Curve points

See also

fitCurvePath()

Implements QwtCurveFitter.

Definition at line 124 of file qwt_weeding_curve_fitter.cpp.

14.172.3.3 fitCurvePath() `QPainterPath QwtWeedingCurveFitter::fitCurvePath ( const QPolygonF & points ) const [override], [virtual]`

Parameters

points	Series of data points

Returns

Curve path

See also

fitCurve()

Implements QwtCurveFitter.

Definition at line 151 of file qwt_weeding_curve_fitter.cpp.

14.172.3.4 setChunkSize() `void QwtWeedingCurveFitter::setChunkSize ( uint numPoints )`

Limit the number of points passed to a run of the algorithm

The runtime of the Douglas Peucker algorithm increases non linear with the number of points. For a chunk size $>$ 0 the polygon is split into pieces passed to the algorithm one by one.

Parameters

numPoints	Maximum for the number of points passed to the algorithm

See also

chunkSize()

Definition at line 101 of file qwt_weeding_curve_fitter.cpp.

14.172.3.5 setTolerance() `void QwtWeedingCurveFitter::setTolerance ( double tolerance )`

Assign the tolerance

The tolerance is the maximum distance, that is acceptable between the original curve and the smoothed curve.

Increasing the tolerance will reduce the number of the resulting points.

Parameters

tolerance	Tolerance

See also

tolerance()

Definition at line 76 of file qwt_weeding_curve_fitter.cpp.

14.172.3.6 tolerance() `double QwtWeedingCurveFitter::tolerance ( ) const`

Returns

Tolerance

See also

setTolerance()

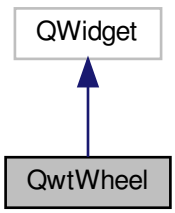
Definition at line 85 of file qwt_weeding_curve_fitter.cpp.

14.173 QwtWheel Class Reference

The Wheel Widget.

`#include <qwt_wheel.h>`

Inheritance diagram for QwtWheel:



Public Slots

- void setValue (double)

  Set a new value without adjusting to the step raster.
- void setTotalAngle (double)

  Set the total angle which the wheel can be turned.
- void setViewAngle (double)

  Specify the visible portion of the wheel.
- void setMass (double)

  Set the slider's mass for flywheel effect.

Signals

- void valueChanged (double value)

  Notify a change of value.
- void wheelPressed ()
- void wheelReleased ()
- void wheelMoved (double value)

Public Member Functions

- QwtWheel (QWidget *parent=NULL)

  Constructor.
- virtual ~QwtWheel ()

  Destructor.
- double value () const
- void setOrientation (Qt::Orientation)

  Set the wheel's orientation.
- Qt::Orientation orientation () const
- double totalAngle () const
- double viewAngle () const
- void setTickCount (int)

  Adjust the number of grooves in the wheel's surface.
- int tickCount () const
- void setWheelWidth (int)

  Set the width of the wheel.
- int wheelWidth () const
- void setWheelBorderWidth (int)

  Set the wheel border width of the wheel.
- int wheelBorderWidth () const
- void setBorderWidth (int)

  Set the border width.
- int borderWidth () const
- void setInverted (bool)

  En/Disable inverted appearance.
- bool isInverted () const
- void setWrapping (bool)

  En/Disable wrapping.
- bool wrapping () const
- void setSingleStep (double)

  Set the step size of the counter.

- double singleStep () const
- void setPageStepCount (int)

  Set the page step count.
- int pageStepCount () const
- void setStepAlignment (bool on)

  En/Disable step alignment.
- bool stepAlignment () const
- void setRange (double min, double max)

  Set the minimum and maximum values.
- void setMinimum (double)
- double minimum () const
- void setMaximum (double)
- double maximum () const
- void setUpdateInterval (int)

  Specify the update interval when the wheel is flying.
- int updateInterval () const
- void setTracking (bool)

  En/Disable tracking.
- bool isTracking () const
- double mass () const

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗) override

  Qt Paint Event.
- virtual void mousePressEvent (QMouseEvent ∗) override

  Mouse press event handler.
- virtual void mouseReleaseEvent (QMouseEvent ∗) override

  Mouse Release Event handler.
- virtual void mouseMoveEvent (QMouseEvent ∗) override

  Mouse Move Event handler.
- virtual void keyPressEvent (QKeyEvent ∗) override
- virtual void wheelEvent (QWheelEvent ∗) override

  Handle wheel events.
- virtual void timerEvent (QTimerEvent ∗) override

  Qt timer event.
- void stopFlying ()

  Stop the flying movement of the wheel.
- QRect wheelRect () const
- virtual QSize sizeHint () const override
- virtual QSize minimumSizeHint () const override
- virtual void drawTicks (QPainter ∗, const QRectF &)
- virtual void drawWheelBackground (QPainter ∗, const QRectF &)
- virtual double valueAt (const QPoint &) const

14.173.1 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 flying wheel.

The default range of the wheel is [0.0, 100.0]

See also

The radio example.

Definition at line 27 of file qwt_wheel.h.

14.173.2 Member Function Documentation

14.173.2.1 borderWidth() `int QwtWheel::borderWidth ( ) const`

Returns

Border width

See also

setBorderWidth()

Definition at line 586 of file qwt_wheel.cpp.

14.173.2.2 drawTicks() `void QwtWheel::drawTicks ( QPainter * painter, const QRectF & rect ) [protected], [virtual]`

Draw the Wheel's ticks

Parameters

painter	Painter
rect	Geometry for the wheel

Definition at line 817 of file qwt_wheel.cpp.

14.173.2.3 drawWheelBackground() `void QwtWheel::drawWheelBackground ( QPainter * painter, const QRectF & rect ) [protected], [virtual]`

Draw the Wheel's background gradient

Parameters

painter	Painter
rect	Geometry for the wheel

Definition at line 761 of file qwt_wheel.cpp.

14.173.2.4 isInverted() `bool QwtWheel::isInverted ( ) const`

Returns

True, when the wheel is inverted

See also

setInverted()

Definition at line 1176 of file qwt_wheel.cpp.

14.173.2.5 isTracking() `bool QwtWheel::isTracking ( ) const`

Returns

True, when tracking is enabled

See also

setTracking(), valueChanged(), wheelMoved()

Definition at line 129 of file qwt_wheel.cpp.

14.173.2.6 keyPressEvent() `void QwtWheel::keyPressEvent ( QKeyEvent * event ) [override], [protected], [virtual]`

Handle key events

- Qt::Key_Home
  Step to minimum()
- Qt::Key_End
  Step to maximum()
- Qt::Key_Up
  In case of a horizontal or not inverted vertical wheel the value will be incremented by the step size. For an inverted vertical wheel the value will be decremented by the step size.
- Qt::Key_Down
  In case of a horizontal or not inverted vertical wheel the value will be decremented by the step size. For an inverted vertical wheel the value will be incremented by the step size.
- Qt::Key_PageUp
  The value will be incremented by pageStepSize() * singleStepSize().
- Qt::Key_PageDown
  The value will be decremented by pageStepSize() * singleStepSize().

Parameters

event	Key event

Definition at line 409 of file qwt_wheel.cpp.

14.173.2.7 mass() `double QwtWheel::mass ( ) const`

Returns

mass

See also

setMass()

Definition at line 1240 of file qwt_wheel.cpp.

14.173.2.8 maximum() `double QwtWheel::maximum ( ) const`

Returns

The maximum of the range

See also

setRange(), setMaximum(), minimum()

Definition at line 1114 of file qwt_wheel.cpp.

14.173.2.9 minimum() `double QwtWheel::minimum ( ) const`

Returns

The minimum of the range

See also

setRange(), setMinimum(), maximum()

Definition at line 1094 of file qwt_wheel.cpp.

14.173.2.10 minimumSizeHint() `QSize QwtWheel::minimumSizeHint ( ) const [override], [protected], [virtual]`

Returns

Minimum size hint

Warning

The return value is based on the wheel width.

Definition at line 962 of file qwt_wheel.cpp.

14.173.2.11 mouseMoveEvent() `void QwtWheel::mouseMoveEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse Move Event handler.

Turn the wheel according to the mouse position

Parameters

event	Mouse event

Definition at line 188 of file qwt_wheel.cpp.

14.173.2.12 mousePressEvent() `void QwtWheel::mousePressEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse press event handler.

Start movement of the wheel.

Parameters

event	Mouse event

Definition at line 163 of file qwt_wheel.cpp.

14.173.2.13 mouseReleaseEvent() `void QwtWheel::mouseReleaseEvent ( QMouseEvent * event ) [override], [protected], [virtual]`

Mouse Release Event handler.

When the wheel has no mass the movement of the wheel stops, otherwise it starts flying.

Parameters

event	Mouse event

Definition at line 237 of file qwt_wheel.cpp.

14.173.2.14 orientation() `Qt::Orientation QwtWheel::orientation ( ) const`

Returns

Orientation

See also

setOrientation()

Definition at line 661 of file qwt_wheel.cpp.

14.173.2.15 pageStepCount() `int QwtWheel::pageStepCount ( ) const`

Returns

Page step count

See also

setPageStepCount(), singleStep()

Definition at line 1043 of file qwt_wheel.cpp.

14.173.2.16 paintEvent() `void QwtWheel::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Qt Paint Event.

Parameters

event	Paint event

Definition at line 736 of file qwt_wheel.cpp.

14.173.2.17 setBorderWidth() `void QwtWheel::setBorderWidth ( int width )`

Set the border width.

The border defaults to 2.

Parameters

width	Border width

See also

borderWidth()

Definition at line 576 of file qwt_wheel.cpp.

14.173.2.18 setInverted() `void QwtWheel::setInverted ( bool on )`

En/Disable inverted appearance.

An inverted wheel increases its values in the opposite direction. The direction of an inverted horizontal wheel will be from right to left an inverted vertical wheel will increase from bottom to top.

Parameters

on	En/Disable inverted appearance

See also

isInverted()

Definition at line 1163 of file qwt_wheel.cpp.

14.173.2.19 setMass `void QwtWheel::setMass ( double mass ) [slot]`

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

mass	New mass in kg

See also

mass()

Definition at line 1221 of file qwt_wheel.cpp.

14.173.2.20 setMaximum() `void QwtWheel::setMaximum ( double value )`

Set the maximum value of the range

Parameters

value	Maximum value

See also

setRange(), setMinimum(), maximum()

Definition at line 1105 of file qwt_wheel.cpp.

14.173.2.21 setMinimum() `void QwtWheel::setMinimum ( double value )`

Set the minimum value of the range

Parameters

value	Minimum value

See also

setRange(), setMaximum(), minimum()

Note

The maximum is adjusted if necessary to ensure that the range remains valid.

Definition at line 1085 of file qwt_wheel.cpp.

14.173.2.22 setOrientation() `void QwtWheel::setOrientation ( Qt::Orientation orientation )`

Set the wheel's orientation.

The default orientation is Qt::Horizontal.

Parameters

orientation	Qt::Horizontal or Qt::Vertical.

See also

orientation()

Definition at line 639 of file qwt_wheel.cpp.

14.173.2.23 setPageStepCount() `void QwtWheel::setPageStepCount ( int count )`

Set the page step count.

pageStepCount is a multiplicator for the single step size that typically corresponds to the user pressing PageUp or PageDown.

A value of 0 disables page stepping.

The default value is 1.

Parameters

count	Multiplicator for the single step size

See also

pageStepCount(), setSingleStep()

Definition at line 1034 of file qwt_wheel.cpp.

14.173.2.24 setRange() `void QwtWheel::setRange (`

```
double min,
double max )
```

Set the minimum and maximum values.

The maximum is adjusted if necessary to ensure that the range remains valid. The value might be modified to be inside of the range.

Parameters

min	Minimum value
max	Maximum value

See also

minimum(), maximum()

Definition at line 1059 of file qwt_wheel.cpp.

14.173.2.25 setSingleStep() `void QwtWheel::setSingleStep (`

```
double stepSize )
```

Set the step size of the counter.

A value <= 0.0 disables stepping

Parameters

stepSize	Single step size

See also

singleStep(), setPageStepCount()

Definition at line 980 of file qwt_wheel.cpp.

14.173.2.26 setStepAlignment() `void QwtWheel::setStepAlignment (`

```
bool on )
```

En/Disable step alignment.

When step alignment is enabled value changes initiated by user input (mouse, keyboard, wheel) are aligned to the multiples of the single step.

Parameters

on	On/Off

See also

stepAlignment(), setSingleStep()

Definition at line 1004 of file qwt_wheel.cpp.

14.173.2.27 setTickCount() `void QwtWheel::setTickCount ( int count )`

Adjust the number of grooves in the wheel's surface.

The number of grooves is limited to 6 <= count <= 50. Values outside this range will be clipped. The default value is 10.

Parameters

count	Number of grooves per 360 degrees

See also

tickCount()

Definition at line 519 of file qwt_wheel.cpp.

14.173.2.28 setTotalAngle `void QwtWheel::setTotalAngle ( double angle ) [slot]`

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*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

angle	total angle in degrees

See also

totalAngle()

Definition at line 613 of file qwt_wheel.cpp.

14.173.2.29 setTracking() `void QwtWheel::setTracking ( bool enable )`

En/Disable tracking.

If tracking is enabled (the default), the wheel emits the valueChanged() signal while the wheel is moving. If tracking is disabled, the wheel emits the valueChanged() signal only when the wheel movement is terminated.

The wheelMoved() signal is emitted regardless id tracking is enabled or not.

Parameters

enable	On/Off

See also

isTracking()

Definition at line 120 of file qwt_wheel.cpp.

14.173.2.30 setUpdateInterval() `void QwtWheel::setUpdateInterval ( int interval )`

Specify the update interval when the wheel is flying.

Default and minimum value is 50 ms.

Parameters

interval	Interval in milliseconds

See also

updateInterval(), setMass(), setTracking()

Definition at line 142 of file qwt_wheel.cpp.

14.173.2.31 setValue `void QwtWheel::setValue ( double value ) [slot]`

Set a new value without adjusting to the step raster.

Parameters

value	New value

See also

value(), valueChanged()

Warning

The value is clipped when it lies outside the range.

Definition at line 1127 of file qwt_wheel.cpp.

14.173.2.32 setViewAngle `void QwtWheel::setViewAngle ( double angle ) [slot]`

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

angle	Visible angle in degrees

See also

viewAngle(), setTotalAngle()

Definition at line 676 of file qwt_wheel.cpp.

14.173.2.33 setWheelBorderWidth() `void QwtWheel::setWheelBorderWidth ( int borderWidth )`

Set the wheel border width of the wheel.

The wheel 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 wheel border defaults to 2.

Parameters

borderWidth	Border width

See also

internalBorder()

Definition at line 551 of file qwt_wheel.cpp.

14.173.2.34 setWheelWidth() `void QwtWheel::setWheelWidth ( int width )`

Set the width of the wheel.

Corresponds to the wheel height for horizontal orientation, and the wheel width for vertical orientation.

Parameters

width	the wheel's width

See also

wheelWidth()

Definition at line 934 of file qwt_wheel.cpp.

14.173.2.35 setWrapping() `void QwtWheel::setWrapping ( bool on )`

En/Disable wrapping.

If wrapping is true stepping up from maximum() value will take you to the minimum() value and vice versa.

Parameters

on	En/Disable wrapping

See also

wrapping()

Definition at line 1190 of file qwt_wheel.cpp.

14.173.2.36 singleStep() `double QwtWheel::singleStep ( ) const`

Returns

Single step size

See also

setSingleStep()

Definition at line 989 of file qwt_wheel.cpp.

14.173.2.37 sizeHint() `QSize QwtWheel::sizeHint ( ) const [override], [protected], [virtual]`

Returns

a size hint

Definition at line 952 of file qwt_wheel.cpp.

14.173.2.38 stepAlignment() `bool QwtWheel::stepAlignment ( ) const`

Returns

True, when the step alignment is enabled

See also

setStepAlignment(), singleStep()

Definition at line 1016 of file qwt_wheel.cpp.

14.173.2.39 tickCount() `int QwtWheel::tickCount ( ) const`

Returns

Number of grooves in the wheel's surface.

See also

setTickCnt()

Definition at line 534 of file qwt_wheel.cpp.

14.173.2.40 timerEvent() `void QwtWheel::timerEvent ( QTimerEvent * event ) [override], [protected], [virtual]`

Qt timer event.

The flying wheel effect is implemented using a timer

Parameters

event	Timer event

See also

updateInterval()

Definition at line 283 of file qwt_wheel.cpp.

14.173.2.41 totalAngle() `double QwtWheel::totalAngle ( ) const`

Returns

Total angle which the wheel can be turned.

See also

setTotalAngle()

Definition at line 626 of file qwt_wheel.cpp.

14.173.2.42 updateInterval() `int QwtWheel::updateInterval ( ) const`

Returns

Update interval when the wheel is flying

See also

setUpdateInterval(), mass(), isTracking()

Definition at line 151 of file qwt_wheel.cpp.

14.173.2.43 value() `double QwtWheel::value ( ) const`

Returns

Current value of the wheel

See also

setValue(), valueChanged()

Definition at line 1147 of file qwt_wheel.cpp.

14.173.2.44 valueAt() `double QwtWheel::valueAt ( const QPoint & pos ) const [protected], [virtual]`

Determine the value corresponding to a specified point

Parameters

pos	Position

Returns

Value corresponding to pos

Definition at line 697 of file qwt_wheel.cpp.

14.173.2.45 valueChanged `void QwtWheel::valueChanged ( double value ) [signal]`

Notify a change of value.

When tracking is enabled this signal will be emitted every time the value changes.

Parameters

value	new value

See also

setTracking()

14.173.2.46 viewAngle() `double QwtWheel::viewAngle ( ) const`

Returns

Visible portion of the wheel

See also

setViewAngle(), totalAngle()

Definition at line 686 of file qwt_wheel.cpp.

14.173.2.47 wheelBorderWidth() `int QwtWheel::wheelBorderWidth ( ) const`

Returns

Wheel border width

See also

setWheelBorderWidth()

Definition at line 563 of file qwt_wheel.cpp.

14.173.2.48 wheelEvent() `void QwtWheel::wheelEvent ( QWheelEvent * event ) [override], [protected], [virtual]`

Handle wheel events.

In/Decrement the value

Parameters

event	Wheel event

Definition at line 324 of file qwt_wheel.cpp.

14.173.2.49 wheelMoved `void QwtWheel::wheelMoved ( double value ) [signal]`

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

Parameters

value	new value

14.173.2.50 wheelPressed `void QwtWheel::wheelPressed ( ) [signal]`

This signal is emitted when the user presses the the wheel with the mouse

14.173.2.51 wheelRect() `QRect QwtWheel::wheelRect ( ) const [protected]`

Returns

Rectangle of the wheel without the outer border

Definition at line 594 of file qwt_wheel.cpp.

14.173.2.52 wheelReleased `void QwtWheel::wheelReleased ( ) [signal]`

This signal is emitted when the user releases the mouse

14.173.2.53 wheelWidth() `int QwtWheel::wheelWidth ( ) const`

Returns

Width of the wheel

See also

setWheelWidth()

Definition at line 944 of file qwt_wheel.cpp.

14.173.2.54 wrapping() `bool QwtWheel::wrapping ( ) const`

Returns

True, when wrapping is set

See also

setWrapping()

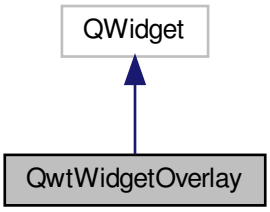
Definition at line 1199 of file qwt_wheel.cpp.

14.174 QwtWidgetOverlay Class Reference

An overlay for a widget.

`#include <qwt_widget_overlay.h>`

Inheritance diagram for QwtWidgetOverlay:



Public Types

- enum MaskMode { NoMask , MaskHint , AlphaMask }
  Mask mode.
- enum RenderMode { AutoRenderMode , CopyAlphaMask , DrawOverlay }
  Render mode.

Public Slots

- void updateOverlay ()

Public Member Functions

- QwtWidgetOverlay (QWidget ∗)
  Constructor.
- virtual ∼QwtWidgetOverlay ()
  Destructor.
- void setMaskMode (MaskMode)
  Specify how to find the mask for the overlay.
- MaskMode maskMode () const
- void setRenderMode (RenderMode)
- RenderMode renderMode () const
- virtual bool eventFilter (QObject ∗, QEvent ∗) override
  Event filter.

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗) override
- virtual void resizeEvent (QResizeEvent ∗) override
- virtual QRegion maskHint () const
  Calculate an approximation for the mask.
- virtual void drawOverlay (QPainter ∗painter) const =0

14.174.1 Detailed Description

An overlay for a widget.

The main use case of an widget overlay is to avoid heavy repaint operation of the widget below.

F.e. in combination with the plot canvas an overlay avoid replots as the content of the canvas can be restored from its backing store.

QwtWidgetOverlay is an abstract base class. Deriving classes are supposed to reimplement the following methods:

- drawOverlay()
- maskHint()

Internally QwtPlotPicker uses overlays for displaying the rubber band and the tracker text.

See also

QwtPlotCanvas::BackingStore

Definition at line 40 of file qwt_widget_overlay.h.

14.174.2 Member Enumeration Documentation

14.174.2.1 MaskMode enum QwtWidgetOverlay::MaskMode

Mask mode.

When using masks the widget below gets paint events for the masked regions of the overlay only. Otherwise Qt triggers full repaints. On less powerful hardware (f.e embedded systems) - or when using the raster paint engine on a remote desktop - bit blitting is a noticeable operation, that needs to be avoided.

If and how to mask depends on how expensive the calculation of the mask is and how many pixels can be excluded by the mask.

The default setting is MaskHint.

See also

setMaskMode(), maskMode()

Enumerator

NoMask	Don't use a mask.
MaskHint	Use maskHint() as mask. For many situations a fast approximation is good enough and it is not necessary to build a more detailed mask (f.e the bounding rectangle of a text).
AlphaMask	Calculate a mask by checking the alpha values. Sometimes it is not possible to give a fast approximation and the mask needs to be calculated by drawing the overlay and testing the result. When a valid maskHint() is available only pixels inside this approximation are checked.

Definition at line 60 of file qwt_widget_overlay.h.

14.174.2.2 RenderMode enum QwtWidgetOverlay::RenderMode

Render mode.

For calculating the alpha mask the overlay has already been painted to a temporary QImage. Instead of rendering the overlay twice this buffer can be copied for drawing the overlay.

On graphic systems using the raster paint engine (QWS, Windows) it means usually copying some memory only. On X11 it results in an expensive operation building a pixmap and for simple overlays it might not be recommended.

Note

The render mode has no effect, when maskMode() != AlphaMask.

Enumerator

AutoRenderMode	Copy the buffer, when using the raster paint engine.
CopyAlphaMask	Always copy the buffer.
DrawOverlay	Never copy the buffer.

Definition at line 102 of file qwt_widget_overlay.h.

14.174.3 Constructor & Destructor Documentation

14.174.3.1 QwtWidgetOverlay() `QwtWidgetOverlay::QwtWidgetOverlay ( QWidget * widget ) [explicit]`

Constructor.

Parameters

widget	Parent widget, where the overlay is aligned to

Definition at line 126 of file qwt_widget_overlay.cpp.

14.174.4 Member Function Documentation

14.174.4.1 drawOverlay() `virtual void QwtWidgetOverlay::drawOverlay ( QPainter * painter ) const [protected], [pure virtual]`

Draw the widget overlay

Parameters

painter	Painter

14.174.4.2 eventFilter() `bool QwtWidgetOverlay::eventFilter ( QObject * object, QEvent * event ) [override], [virtual]`

Event filter.

Resize the overlay according to the size of the parent widget.

Parameters

object	Object to be filtered
event	Event

Returns

See QObject::eventFilter()

Definition at line 389 of file qwt_widget_overlay.cpp.

14.174.4.3 maskHint() `QRegion QwtWidgetOverlay::maskHint ( ) const [protected], [virtual]`

Calculate an approximation for the mask.

- MaskHint The hint is used as mask.
- AlphaMask The hint is used to speed up the algorithm for calculating a mask from non transparent pixels
- NoMask The hint is unused.

The default implementation returns an invalid region indicating no hint.

Returns

Hint for the mask

Definition at line 373 of file qwt_widget_overlay.cpp.

14.174.4.4 maskMode() `QwtWidgetOverlay::MaskMode QwtWidgetOverlay::maskMode ( ) const`

Returns

Mode how to find the mask for the overlay

See also

setMaskMode()

Definition at line 167 of file qwt_widget_overlay.cpp.

14.174.4.5 paintEvent() `void QwtWidgetOverlay::paintEvent ( QPaintEvent * event ) [override], [protected], [virtual]`

Paint event

Parameters

event	Paint event

See also

drawOverlay()

Definition at line 261 of file qwt_widget_overlay.cpp.

14.174.4.6 renderMode() `QwtWidgetOverlay::RenderMode QwtWidgetOverlay::renderMode ( ) const`

Returns

Render mode

See also

RenderMode, setRenderMode()

Definition at line 187 of file qwt_widget_overlay.cpp.

14.174.4.7 resizeEvent() `void QwtWidgetOverlay::resizeEvent ( QResizeEvent * event ) [override], [protected], [virtual]`

Resize event

Parameters

event	Resize event

Definition at line 323 of file qwt_widget_overlay.cpp.

14.174.4.8 setMaskMode() `void QwtWidgetOverlay::setMaskMode ( MaskMode mode )`

Specify how to find the mask for the overlay.

Parameters

mode	New mode

See also

maskMode()

Definition at line 154 of file qwt_widget_overlay.cpp.

14.174.4.9 setRenderMode() void QwtWidgetOverlay::setRenderMode (RenderMode *mode*)

Set the render mode

Parameters

mode	Render mode

See also

RenderMode, renderMode()

Definition at line 178 of file qwt_widget_overlay.cpp.

14.174.4.10 updateOverlay void QwtWidgetOverlay::updateOverlay ( ) [slot]

Recalculate the mask and repaint the overlay

Definition at line 195 of file qwt_widget_overlay.cpp.

Index