Qwt User's Guide

5.2.1

Generated by Doxygen 1.6.3

Sun Apr 11 11:55:10 2010

Contents

1 Qwt - Qt Widgets for Technical Applications

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

1.1 License

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

1.2 Platforms

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

1.3 Screenshots

- Curve Plots
- Scatter Plot
- Spectrogram, Contour Plot
- Histogram
- Dials, Compasses, Knobs, Wheels, Sliders, Thermos

Screenshots are only available in the HTML docs.

1.4 Downloads

Stable releases, prereleases and snapshots are available at the Qwt project page.
For getting a snapshot with all bugfixes for the latest 5.2 release:

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

For getting a development snapshot from the SVN repository:

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

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

1.5 Installation

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

1.6 Support

- Mailing list

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

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

- Forum

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

- Individual support

  If you are looking for individual support, or need someone who implements your Qwt component/application contact qwt-support@tigertal.de.

1.7 Related Projects

QwtPolar, a polar plot widget.

QwtPlot3D, an OpenGL 3D plot widget.

QtiPlot, data analysis and scientific plotting tool, using QwtPlot.

1.8 Language Bindings

PyQwt, a set of Qwt Python bindings.

Korundum/QtRuby, including a set of Qwt Ruby bindings.

1.9 Donations

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

1.10 Credits:

Authors:

Uwe Rathmann, Josef Wilgen (<= Qwt 0.2)

Project admin:

Uwe Rathmann <rathmann@users.sourceforge.net>

2 Qwt License, Version 1.0

```
                            Qwt License
                        Version 1.0, January 1, 2003

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

    1. Widgets that are subclassed from Qwt widgets do not
       constitute a derivative work.

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

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

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


       However, programs must still identify their use of Qwt.
       The following example statement can be included in user
       documentation to satisfy this requirement:

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

------------------------------------------------------------------------


          GNU LESSER GENERAL PUBLIC LICENSE
               Version 2.1, February 1999

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

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

                Preamble

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

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

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

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

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

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

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

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

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

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

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

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

case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. You may copy and distribute the Library (or a portion or

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

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

5. A program that contains no derivative of any portion of the
Library, but is designed to work with the Library by being compiled or
linked with it, is called a "work that uses the Library". Such a
work, in isolation, is not a derivative work of the Library, and
therefore falls outside the scope of this License.

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

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

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

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

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

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

a) Accompany the work with the complete corresponding
machine-readable source code for the Library including whatever
changes were used in the work (which must be distributed under
Sections 1 and 2 above); and, if the work is an executable linked
with the Library, with the complete machine-readable "work that
uses the Library", as object code and/or source code, so that the
user can modify the Library and then relink to produce a modified
executable containing the modified Library. (It is understood
that the user who changes the contents of definitions files in the

```
    Library will not necessarily be able to recompile the application
    to use the modified definitions.)

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

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

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

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

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

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

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

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

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

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

  9. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Library or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Library (or any work based on the
```

Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

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

11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License. If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all. For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

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

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

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

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

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

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

14. If you wish to incorporate parts of the Library into other free
programs whose distribution conditions are incompatible with these,
write to the author to ask for permission. For software which is

copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this. Our
decision will be guided by the two goals of preserving the free status
of all derivatives of our free software and of promoting the sharing
and reuse of software generally.

NO WARRANTY

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

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

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

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

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

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

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

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

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

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

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

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

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

That's all there is to it!
```

3 INSTALL

```
Introduction
============

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

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

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

MathML Extension
================

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

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

Documentation
=============================

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

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

A) Unix Qt3/Qt4
===========================

qmake
make
make install

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

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

If you didn't enable autobuilding of the examples in qwtconfig.pri
```

```
you have to build the examples this way:

cd examples
qmake
make


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

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

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

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

cd examples
qmake examples.pro
nmake

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

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

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

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


C) Win32/MinGW Qt4
==================

C1) Windows Shell

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

qmake qwt.pro
make

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

cd examples
qmake examples.pro
make
make install

C2) MSYS Shell Qt >= 4.3.0

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

C3) MSYS Shell Qt < 4.3.0
```

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

export MINGW_IN_SHELL=1;

qmake
make -i
make -i install

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

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

C1-C3)

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

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

D) MacOSX

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

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

qmake -spec macx-g++
...

D) Qtopia Core

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

qmake
make

E) Qtopia (!= Qtopia Core)

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

Good luck !
```

4 Curve Plots

5 Scatter Plot

6 Spectrogram, Contour Plot

/∗!

7 Histogram

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

9 Deprecated List

Member QwtPlot::clear() Use QwtPlotDeict::detachItems instead

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

10 Class Index

10.1 Class Hierarchy

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

11 Class Index

11.1 Class List

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

12 Class Documentation

12.1 QwtEventPattern::KeyPattern Class Reference

A pattern for key events.

`#include <qwt_event_pattern.h>`

Public Member Functions

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

Public Attributes

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

12.1.1 Detailed Description

A pattern for key events.

12.2 QwtEventPattern::MousePattern Class Reference

A pattern for mouse events.

`#include <qwt_event_pattern.h>`

Public Member Functions

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

Public Attributes

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

12.2.1 Detailed Description

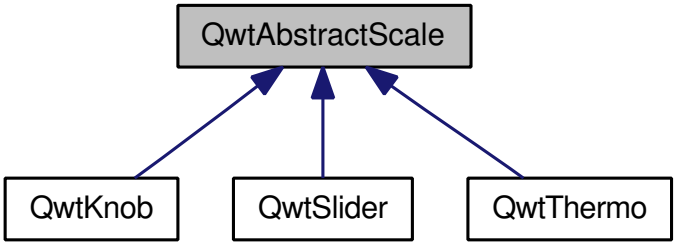
A pattern for mouse events.

12.3 QwtAbstractScale Class Reference

An abstract base class for classes containing a scale.

`#include <qwt_abstract_scale.h>`

Inheritance diagram for QwtAbstractScale:



Public Member Functions

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

Protected Member Functions

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

12.3.1 Detailed Description

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

12.3.2 Constructor & Destructor Documentation

12.3.2.1 QwtAbstractScale::QwtAbstractScale ()

Constructor

Creates a default QwtScaleDraw and a QwtLinearScaleEngine. Autoscaling is enabled, and the stepSize is initialized by 0.0.

12.3.2.2 QwtAbstractScale::~QwtAbstractScale () [virtual]

Destructor.

12.3.3 Member Function Documentation

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

Returns

Scale draw

See also

setAbstractScaleDraw()

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

Returns

Scale draw

See also

setAbstractScaleDraw()

12.3.3.3 bool QwtAbstractScale::autoScale () const

Returns

`true` if autoscaling is enabled

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

Recalculate the scale division and update the scale draw.

Parameters

vmin Lower limit of the scale interval

vmax Upper limit of the scale interval

stepSize Major step size

See also

scaleChange()

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

Notify changed scale.

Dummy empty implementation, intended to be overloaded by derived classes

Reimplemented in QwtSlider, and QwtThermo.

12.3.3.6 QwtScaleEngine * QwtAbstractScale::scaleEngine ()

Returns

Scale engine

See also

setScaleEngine()

12.3.3.7 const QwtScaleEngine * QwtAbstractScale::scaleEngine () const

Returns

Scale engine

See also

setScaleEngine()

12.3.3.8 const QwtScaleMap & QwtAbstractScale::scaleMap () const

Returns

abstractScaleDraw()->scaleMap()

12.3.3.9 int QwtAbstractScale::scaleMaxMajor () const

Returns

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

12.3.3.10 int QwtAbstractScale::scaleMaxMinor () const

Returns

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

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

Set a scale draw.

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

12.3.3.12 void QwtAbstractScale::setAutoScale ()

Advise the widget to control the scale range internally.

Autoscaling is on by default.

See also

setScale(), autoScale()

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

Specify a scale.

Disable autoscaling and define a scale by a scale division

Parameters

scaleDiv Scale division

See also

setAutoScale()

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

Specify a scale.

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

Parameters

interval Interval

stepSize major step size

See also

setAutoScale()

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

Specify a scale.

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

Parameters

vmin lower limit of the scale interval

vmax upper limit of the scale interval

stepSize major step size

See also

setAutoScale()

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

Set a scale engine.

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

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

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

Set the maximum number of major tick intervals.

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

Parameters

ticks maximal number of major ticks.

See also

QwtAbstractScaleDraw

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

Set the maximum number of minor tick intervals.

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

Parameters

ticks

See also

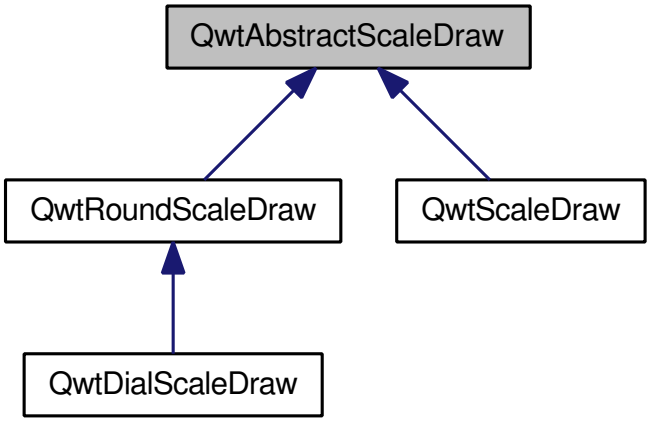
QwtAbstractScaleDraw

12.4 QwtAbstractScaleDraw Class Reference

A abstract base class for drawing scales.

`#include <qwt_abstract_scale_draw.h>`

Inheritance diagram for QwtAbstractScaleDraw:



Public Types

- enum ScaleComponent {
  Backbone = 1,
  Ticks = 2,
  Labels = 4 }

Public Member Functions

- QwtAbstractScaleDraw ()
- QwtAbstractScaleDraw (const QwtAbstractScaleDraw &)
- virtual ~QwtAbstractScaleDraw ()
- QwtAbstractScaleDraw & operator= (const QwtAbstractScaleDraw &)
- void setScaleDiv (const QwtScaleDiv &s)
- const QwtScaleDiv & scaleDiv () const
- void setTransformation (QwtScaleTransformation *)
- const QwtScaleMap & map () const
- void enableComponent (ScaleComponent, bool enable=true)
- bool hasComponent (ScaleComponent) const
- void setTickLength (QwtScaleDiv::TickType, int length)

- int tickLength (QwtScaleDiv::TickType) const
- int majTickLength () const
- void setSpacing (int margin)
- int spacing () const
- virtual void draw (QPainter *, const QPalette &) const
- virtual QwtText label (double) const
- virtual int extent (const QPen &, const QFont &) const =0
- void setMinimumExtent (int)
- int minimumExtent () const
- QwtScaleMap & scaleMap ()

Protected Member Functions

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

12.4.1 Detailed Description

A abstract base class for drawing scales. QwtAbstractScaleDraw can be used to draw linear or logarithmic scales.

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

12.4.2 Member Enumeration Documentation

12.4.2.1 enum QwtAbstractScaleDraw::ScaleComponent

Components of a scale

- Backbone
- Ticks
- Labels

See also

enableComponent(), hasComponent

12.4.3 Constructor & Destructor Documentation

12.4.3.1 QwtAbstractScaleDraw::QwtAbstractScaleDraw ()

Constructor.

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

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

Copy constructor.

12.4.3.3 QwtAbstractScaleDraw::~QwtAbstractScaleDraw () [virtual]

Destructor.

12.4.4 Member Function Documentation

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

Draw the scale.

Parameters

painter The painter

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

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

Draws the baseline of the scale

Parameters

painter Painter

See also

drawTick(), drawLabel()

Implemented in QwtRoundScaleDraw, and QwtScaleDraw.

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

Draws the label for a major scale tick

Parameters

painter Painter

value Value

See also

drawTick, drawBackbone

Implemented in QwtRoundScaleDraw, and QwtScaleDraw.

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

Draw a tick

Parameters

painter Painter

value Value of the tick

len Lenght of the tick

See also

drawBackbone(), drawLabel()

Implemented in QwtRoundScaleDraw, and QwtScaleDraw.

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

En/Disable a component of the scale

Parameters

component Scale component

enable On/Off

See also

hasComponent()

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

Calculate the extent

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

See also

setMinimumExtent(), minimumExtent()

Implemented in QwtRoundScaleDraw, and QwtScaleDraw.

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

Check if a component is enabled

See also

enableComponent()

12.4.4.8 void QwtAbstractScaleDraw::invalidateCache () [protected]

Invalidate the cache used by QwtAbstractScaleDraw::tickLabel

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

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

Convert a value into its representing label.

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

Parameters

value Value

Returns

Label string.

Reimplemented in QwtDialScaleDraw.

12.4.4.10 int QwtAbstractScaleDraw::majTickLength () const

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

12.4.4.11 const QwtScaleMap & QwtAbstractScaleDraw::map () const

Returns

Map how to translate between scale and pixel values

12.4.4.12 int QwtAbstractScaleDraw::minimumExtent () const

Get the minimum extent

See also

extent(), setMinimumExtent()

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

Assignment operator.

12.4.4.14 const QwtScaleDiv & QwtAbstractScaleDraw::scaleDiv () const

Returns

scale division

12.4.4.15 QwtScaleMap & QwtAbstractScaleDraw::scaleMap ()

Returns

Map how to translate between scale and pixel values

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

Set a minimum for the extent.

The extent is calculated from the coomponents 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()

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

Change the scale division

Parameters

sd New scale division

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

Set the spacing between tick and labels.

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

Parameters

spacing Spacing

See also

spacing()

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

Set the length of the ticks

Parameters

tickType Tick type

length New length

Warning

the length is limited to [0..1000]

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

Change the transformation of the scale

Parameters

transformation New scale transformation

12.4.4.21 int QwtAbstractScaleDraw::spacing () const

Get the spacing.

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

See also

setSpacing()

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

Convert a value into its representing label and cache it.

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

Parameters

font Font

value Value

Returns

Tick label

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

Return the length of the ticks

See also

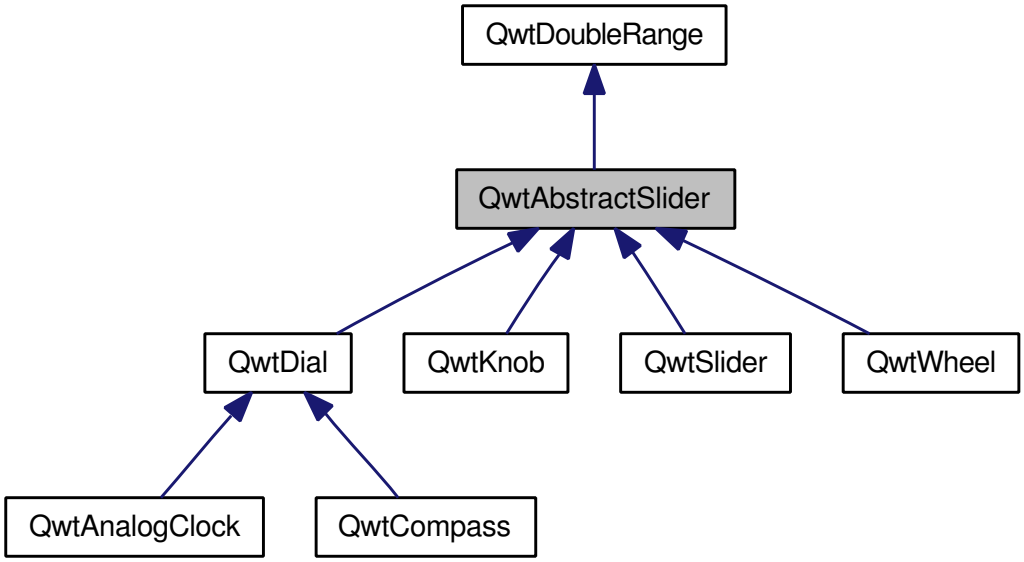
setTickLength(), majTickLength()

12.5 QwtAbstractSlider Class Reference

An abstract base class for slider widgets.

`#include <qwt_abstract_slider.h>`

Inheritance diagram for QwtAbstractSlider:



Public Types

- enum ScrollMode {
  ScrNone,
  ScrMouse,
  ScrTimer,
  ScrDirect,
  ScrPage }

Public Slots

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

Signals

- void valueChanged (double value)
- void sliderPressed ()
- void sliderReleased ()
- void sliderMoved (double value)

Public Member Functions

- QwtAbstractSlider (Qt::Orientation, QWidget *parent=NULL)
- virtual ~QwtAbstractSlider ()
- void setUpdateTime (int t)
- void stopMoving ()
- void setTracking (bool enable)
- virtual void setMass (double val)
- virtual double mass () const
- virtual void setOrientation (Qt::Orientation o)
- Qt::Orientation orientation () const
- bool isReadOnly () const
- bool isValid () const
- void setValid (bool valid)

Protected Member Functions

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

12.5.1 Detailed Description

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

12.5.2 Member Enumeration Documentation

12.5.2.1 enum QwtAbstractSlider::ScrollMode

Scroll mode

See also

getScrollMode()

12.5.3 Constructor & Destructor Documentation

12.5.3.1 QwtAbstractSlider::QwtAbstractSlider (Qt::Orientation *orientation*, QWidget * *parent* = NULL) [explicit]

Constructor.

Parameters

orientation Orientation

parent Parent widget

12.5.3.2 QwtAbstractSlider::~QwtAbstractSlider () [virtual]

Destructor.

12.5.4 Member Function Documentation

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

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

Parameters

value Value

See also

setValue(), incValue()

Reimplemented from QwtDoubleRange.

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

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

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

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

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

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

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

Parameters

p point where the mouse was pressed

Return values

scrollMode The scrolling mode

direction direction: 1, 0, or -1.

Implemented in QwtDial, QwtSlider, and QwtWheel.

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

Determine the value corresponding to a specified poind.

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

Parameters

p point

Implemented in QwtDial, QwtSlider, and QwtWheel.

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

Increment the value by a specified number of steps.

Parameters

steps number of steps

See also

setValue()

Reimplemented from QwtDoubleRange.

12.5.4.5 bool QwtAbstractSlider::isReadOnly () const

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

Returns

true if read only

See also

setReadOnly()

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

See also

QwtDblRange::isValid()

Reimplemented from QwtDoubleRange.

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

Handles key events

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

Parameters

e Key event

See also

isReadOnly()

Reimplemented in QwtCompass, and QwtDial.

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

Returns

mass

See also

setMass()

Reimplemented in QwtWheel.

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

Mouse Move Event handler

Parameters

e Mouse event

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

Mouse press event handler

Parameters

e Mouse event

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

Mouse Release Event handler

Parameters

e Mouse event

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

Returns

Orientation

See also

setOrientation()

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

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

val New mass in kg

See also

mass()

Reimplemented in QwtWheel.

12.5.4.14 void QwtAbstractSlider::setOrientation (Qt::Orientation *o*) [virtual]

Set the orientation.

Parameters

o Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.

Reimplemented in QwtSlider, and QwtWheel.

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

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

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

En/Disable read only mode

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

Parameters

readOnly Enables in case of true

See also

isReadOnly()

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

Enables or disables tracking.

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

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

Tracking is enabled by default.

Parameters

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

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

Specify the update interval for automatic scrolling.

Parameters

t update interval in milliseconds

See also

getScrollMode()

12.5.4.19 void QwtAbstractSlider::setValid (bool *valid*) `[inline]`

Parameters

valid true/false

See also

QwtDblRange::isValid()

Reimplemented from QwtDoubleRange.

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

Move the slider to a specified value.

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

Parameters

val new value

See also

fitValue()

Reimplemented from QwtDoubleRange.

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

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

Parameters

value new value

12.5.4.22 void QwtAbstractSlider::sliderPressed () [signal]

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

12.5.4.23 void QwtAbstractSlider::sliderReleased () [signal]

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

12.5.4.24 void QwtAbstractSlider::stopMoving ()

Stop updating if automatic scrolling is active.

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

Qt timer event

Parameters

e Timer event

12.5.4.26 void QwtAbstractSlider::valueChange () [protected, virtual]

Notify change of value

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

Reimplemented from QwtDoubleRange.

Reimplemented in QwtDial, QwtSlider, and QwtWheel.

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

Notify a change of value.

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

Parameters

value new value

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

Wheel Event handler

Parameters

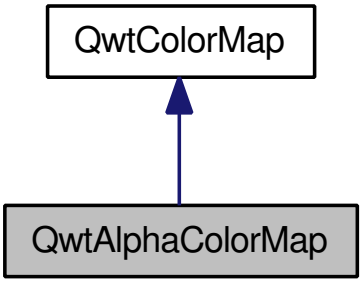
e Whell event

12.6 QwtAlphaColorMap Class Reference

QwtAlphaColorMap variies the alpha value of a color.

`#include <qwt_color_map.h>`

Inheritance diagram for QwtAlphaColorMap:



Public Member Functions

- QwtAlphaColorMap (const QColor &=QColor(Qt::gray))
- QwtAlphaColorMap (const QwtAlphaColorMap &)
- virtual ~QwtAlphaColorMap ()
- QwtAlphaColorMap & operator= (const QwtAlphaColorMap &)
- virtual QwtColorMap * copy () const
- void setColor (const QColor &)
- QColor color () const
- virtual QRgb rgb (const QwtDoubleInterval &, double value) const

12.6.1 Detailed Description

QwtAlphaColorMap variies the alpha value of a color.

12.6.2 Constructor & Destructor Documentation

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

Constructor

Parameters

color Color of the map

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

Copy constructor

Parameters

other Other color map

12.6.2.3 QwtAlphaColorMap::~QwtAlphaColorMap () [virtual]

Destructor.

12.6.3 Member Function Documentation

12.6.3.1 QColor QwtAlphaColorMap::color () const

Returns

the color

See also

setColor()

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

Clone the color map.

Implements QwtColorMap.

12.6.3.3 QwtAlphaColorMap & QwtAlphaColorMap::operator= (const QwtAlphaColorMap & *other*)

Assignment operator

Parameters

other Other color map

Returns

*this

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

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

alpha := (value - interval.minValue()) / interval.width();

Parameters

interval Range for all values

value Value to map into a rgb value

Returns

rgb value, with an alpha value

Implements QwtColorMap.

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

Set the color

Parameters

color Color

See also

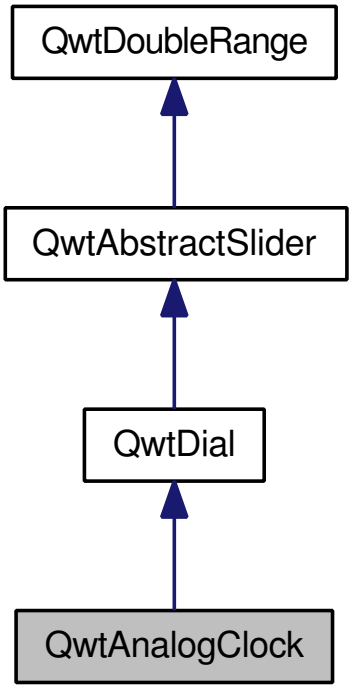
color()

12.7 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 ()
- void setTime (const QTime &=QTime::currentTime())

Public Member Functions

- QwtAnalogClock (QWidget *parent=NULL)
- virtual ~QwtAnalogClock ()
- virtual void setHand (Hand, QwtDialNeedle *)
- const QwtDialNeedle * hand (Hand) const
- QwtDialNeedle * hand (Hand)

Protected Member Functions

- virtual QwtText scaleLabel (double) const
- virtual void drawNeedle (QPainter *, const QPoint &, int radius, double direction, QPalette::ColorGroup) const
- virtual void drawHand (QPainter *, Hand, const QPoint &, int radius, double direction, QPalette::ColorGroup) const

12.7.1 Detailed Description

An analog clock.

Example

```
#include <qwt_analog_clock.h>

  QwtAnalogClock *clock = new QwtAnalogClock(...);
  clock->scaleDraw()->setPenWidth(3);
  clock->setLineWidth(6);
  clock->setFrameShadow(QwtDial::Sunken);
  clock->setTime();

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

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

Note

The examples/dials example shows how to use QwtAnalogClock.

12.7.2 Member Enumeration Documentation

12.7.2.1 enum QwtAnalogClock::Hand

Hand type

See also

setHand(), hand()

12.7.3 Constructor & Destructor Documentation

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

Constructor

Parameters

parent Parent widget

12.7.3.2 QwtAnalogClock::~QwtAnalogClock () [virtual]

Destructor.

12.7.4 Member Function Documentation

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

Draw a clock hand

Parameters

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

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

Draw the needle.

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

Parameters

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

See also

drawHand()

Reimplemented from QwtDial.

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

Returns

Clock hand

Parameters

hd Specifies the type of hand

See also

setHand()

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

Returns

Clock hand

Parameters

hd Specifies the type of hand

See also

setHand()

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

Find the scale label for a given value

Parameters

value Value

Returns

Label

Reimplemented from QwtDial.

12.7.4.6 void QwtAnalogClock::setCurrentTime () [slot]

Set the current time.

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

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

Set a clockhand

Parameters

hand Specifies the type of hand

needle Hand

See also

hand()

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

Set a time

Parameters

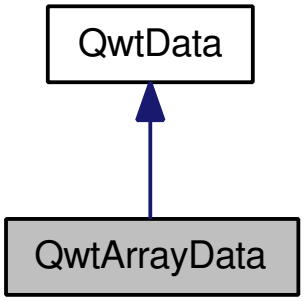
time Time to display

12.8 QwtArrayData Class Reference

Data class containing two QwtArray<double> objects.

`#include <qwt_data.h>`

Inheritance diagram for QwtArrayData:



Public Member Functions

- QwtArrayData (const QwtArray< double > &x, const QwtArray< double > &y)
- QwtArrayData (const double *x, const double *y, size_t size)

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

12.8.1 Detailed Description

Data class containing two QwtArray<double> objects.

12.8.2 Constructor & Destructor Documentation

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

Constructor

Parameters

x Array of x values

y Array of y values

See also

QwtPlotCurve::setData()

12.8.2.2 QwtArrayData::QwtArrayData (const double * *x*, const double * *y*, size_t *size*)

Constructor

Parameters

x Array of x values

y Array of y values

size Size of the x and y arrays

See also

QwtPlotCurve::setData()

12.8.3 Member Function Documentation

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

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

Reimplemented from QwtData.

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

Returns

Pointer to a copy (virtual copy constructor)

Implements QwtData.

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

Assignment.

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

Returns

Size of the data set

Implements QwtData.

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

Return the x value of data point i

Parameters

i Index

Returns

x X value of data point i

Implements QwtData.

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

Returns

Array of the x-values

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

Return the y value of data point i

Parameters

i Index

Returns

y Y value of data point i

Implements QwtData.

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

Returns

Array of the y-values

12.9 QwtArrowButton Class Reference

Arrow Button.

`#include <qwt_arrow_button.h>`

Public Member Functions

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

Protected Member Functions

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

12.9.1 Detailed Description

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

12.9.2 Constructor & Destructor Documentation

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

Parameters

num Number of arrows

arrowType see Qt::ArowType in the Qt docs.

parent Parent widget

12.9.2.2 QwtArrowButton::~QwtArrowButton () [virtual]

Destructor.

12.9.3 Member Function Documentation

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

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

Parameters

arrowType Arrow type

boundingSize Bounding size

Returns

Size of the arrow

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

The direction of the arrows.

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

Draw an arrow int a bounding rect

Parameters

painter Painter

r Rectangle where to paint the arrow

arrowType Arrow type

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

Draw the button label.

Parameters

painter Painter

See also

The Qt Manual on QPushButton

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

autoRepeat for the space keys

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

Returns

the bounding rect for the label

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

Return a minimum size hint.

12.9.3.8 int QwtArrowButton::num () const

The number of arrows.

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

Paint event handler

Parameters

event Paint event

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

Returns

a size hint

12.10 QwtClipper Class Reference

Some clipping algos.

`#include <qwt_clipper.h>`

Static Public Member Functions

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

12.10.1 Detailed Description

Some clipping algos.

12.10.2 Member Function Documentation

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

Circle clipping

clipCircle() devides 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

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

Sutherland-Hodgman polygon clipping

Parameters

clipRect Clip rectangle

polygon Polygon

Returns

Clipped polygon

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

Sutherland-Hodgman polygon clipping

Parameters

clipRect Clip rectangle

polygon Polygon

Returns

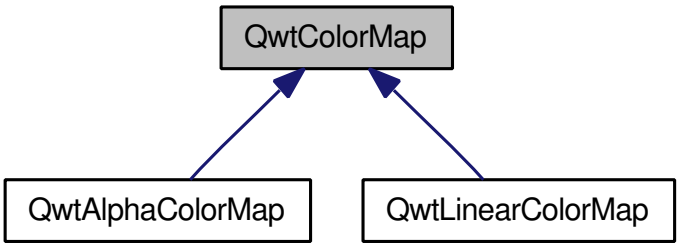
Clipped polygon

12.11 QwtColorMap Class Reference

QwtColorMap is used to map values into colors.

`#include <qwt_color_map.h>`

Inheritance diagram for QwtColorMap:



Public Types

- enum Format {
  RGB,
  Indexed }

Public Member Functions

- QwtColorMap (Format=QwtColorMap::RGB)
- virtual ~QwtColorMap ()
- Format format () const
- virtual QwtColorMap * copy () const =0
- virtual QRgb rgb (const QwtDoubleInterval &interval, double value) const =0

- virtual unsigned char colorIndex (const QwtDoubleInterval &interval, double value) const =0
- QColor color (const QwtDoubleInterval &, double value) const
- virtual QVector< QRgb > colorTable (const QwtDoubleInterval &) const

12.11.1 Detailed Description

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

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

- QImage::Format_Indexed8
- QImage::Format_ARGB32

See also

QwtPlotSpectrogram, QwtScaleWidget

12.11.2 Member Enumeration Documentation

12.11.2.1 enum QwtColorMap::Format

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

See also

rgb(), colorIndex(), colorTable()

12.11.3 Constructor & Destructor Documentation

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

Constructor.

12.11.3.2 QwtColorMap::~QwtColorMap () `[virtual]`

Destructor.

12.11.4 Member Function Documentation

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

Map a value into a color

Parameters

interval Valid interval for values

value Value

Returns

Color corresponding to value

Warning

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

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

Map a value of a given interval into a color index

Parameters

interval Range for the values

value Value

Returns

color index, corresponding to value

Implemented in QwtLinearColorMap.

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

Build and return a color map of 256 colors

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

Parameters

interval Range for the values

Returns

A color table, that can be used for a QImage

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

Clone the color map.

Implemented in QwtLinearColorMap, and QwtAlphaColorMap.

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

Returns

Intended format of the color map

See also

Format

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

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

Parameters

interval Range for the values

value Value

Returns

rgb value, corresponding to value

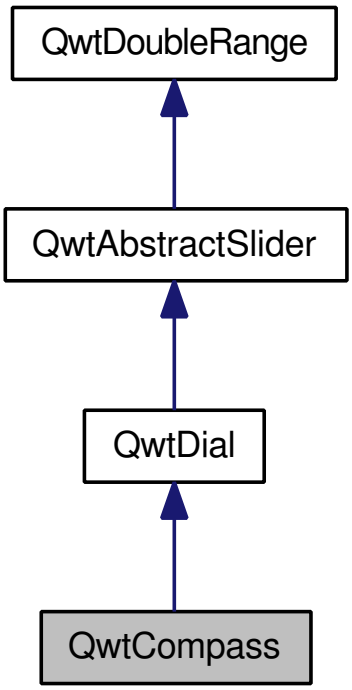
Implemented in QwtLinearColorMap, and QwtAlphaColorMap.

12.12 QwtCompass Class Reference

A Compass Widget.

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

Inheritance diagram for QwtCompass:



Public Member Functions

- QwtCompass (QWidget *parent=NULL)
- virtual ~QwtCompass ()
- void setRose (QwtCompassRose *rose)
- const QwtCompassRose * rose () const
- QwtCompassRose * rose ()
- const QMap< double, QString > & labelMap () const
- QMap< double, QString > & labelMap ()
- void setLabelMap (const QMap< double, QString > &map)

Protected Member Functions

- virtual QwtText scaleLabel (double value) const
- virtual void drawRose (QPainter *, const QPoint &center, int radius, double north, QPalette::ColorGroup) const
- virtual void drawScaleContents (QPainter *, const QPoint &center, int radius) const
- virtual void keyPressEvent (QKeyEvent *)

12.12.1 Detailed Description

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

Note

The examples/dials example shows how to use QwtCompass.

12.12.2 Constructor & Destructor Documentation

12.12.2.1 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.

12.12.2.2 QwtCompass::~QwtCompass () [virtual]

Destructor.

12.12.3 Member Function Documentation

12.12.3.1 void QwtCompass::drawRose (QPainter * *painter*, const QPoint & *center*, int *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

12.12.3.2 void QwtCompass::drawScaleContents (QPainter * *painter*, const QPoint & *center*, int *radius*) const [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.

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

Handles key events

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

See also

isReadOnly()

Reimplemented from QwtDial.

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

Returns

map, mapping values to labels

See also

setLabelMap()

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

Returns

map, mapping values to labels

See also

setLabelMap()

12.12.3.6 QwtCompassRose * QwtCompass::rose ()

Returns

rose

See also

setRose()

12.12.3.7 const QwtCompassRose * QwtCompass::rose () const

Returns

rose

See also

setRose()

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

Map a value to a corresponding label

Parameters

value Value that will be mapped

Returns

Label, or QString::null

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

See also

labelMap(), setLabelMap()

Reimplemented from QwtDial.

12.12.3.9 void QwtCompass::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()

12.12.3.10 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

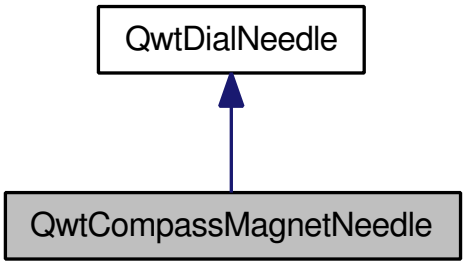
rose()

12.13 QwtCompassMagnetNeedle Class Reference

A magnet needle for compass widgets.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtCompassMagnetNeedle:



Public Types

- enum Style {
  TriangleStyle,
  ThinStyle }

Public Member Functions

- QwtCompassMagnetNeedle (Style=TriangleStyle, const QColor &light=Qt::white, const QColor &dark=Qt::red)
- virtual void draw (QPainter *, const QPoint &, int length, double direction, QPalette::ColorGroup=QPalette::Active) const

Static Public Member Functions

- static void drawTriangleNeedle (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)
- static void drawThinNeedle (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)

Static Protected Member Functions

- static void drawPointer (QPainter *painter, const QBrush &brush, int colorOffset, const QPoint &center, int length, int width, double direction)

12.13.1 Detailed Description

A magnet needle for compass widgets. A magnet needle points to two opposite directions indicating north and south.

The following colors are used:

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

See also

QwtDial, QwtCompass

12.13.2 Member Enumeration Documentation

12.13.2.1 enum QwtCompassMagnetNeedle::Style

Style of the needle.

12.13.3 Constructor & Destructor Documentation

12.13.3.1 QwtCompassMagnetNeedle::QwtCompassMagnetNeedle (Style *style* = `TriangleStyle`, const QColor & *light* = `Qt::white`, const QColor & *dark* = `Qt::red`)

Constructor.

12.13.4 Member Function Documentation

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

Draw the needle

Parameters

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

Implements QwtDialNeedle.

12.13.4.2 void QwtCompassMagnetNeedle::drawPointer (QPainter * *painter*, const QBrush & *brush*, int *colorOffset*, const QPoint & *center*, int *length*, int *width*, double *direction*) [static, protected]

Draw a compass needle

Parameters

painter Painter

brush Brush

colorOffset Color offset

center Center, where the needle starts

length Length of the needle

width Width of the needle

direction Direction

12.13.4.3 void QwtCompassMagnetNeedle::drawThinNeedle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, double *direction*) [static]

Draw a compass needle

Parameters

painter Painter

palette Palette

colorGroup Color group

center Center, where the needle starts

length Length of the needle

direction Direction

12.13.4.4 void QwtCompassMagnetNeedle::drawTriangleNeedle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, double *direction*) [static]

Draw a compass needle

Parameters

painter Painter

palette Palette

colorGroup Color group

center Center, where the needle starts

length Length of the needle

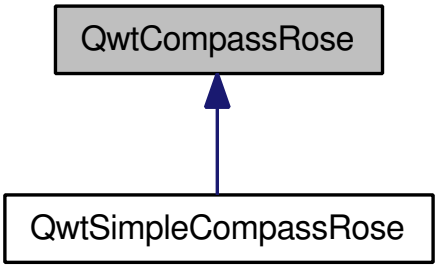
direction Direction

12.14 QwtCompassRose Class Reference

Abstract base class for a compass rose.

`#include <qwt_compass_rose.h>`

Inheritance diagram for QwtCompassRose:



Public Member Functions

- virtual void setPalette (const QPalette &p)
- const QPalette & palette () const
- virtual void draw (QPainter *painter, const QPoint &center, int radius, double north, QPalette::ColorGroup colorGroup=QPalette::Active) const =0

12.14.1 Detailed Description

Abstract base class for a compass rose.

12.14.2 Member Function Documentation

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

Draw the rose

Parameters

painter Painter

center Center point

radius Radius of the rose

north Position

colorGroup Color group

Implemented in QwtSimpleCompassRose.

12.14.2.2 const QPalette& QwtCompassRose::palette () const [inline]

Returns

Current palette

12.14.2.3 virtual void QwtCompassRose::setPalette (const QPalette & *p*) [inline, virtual]

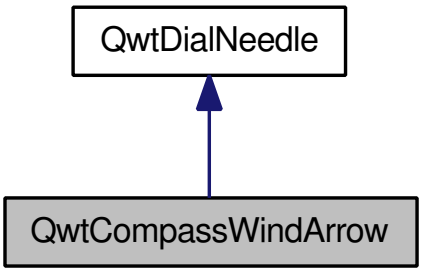
Assign a palette.

12.15 QwtCompassWindArrow Class Reference

An indicator for the wind direction.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtCompassWindArrow:



Public Types

- enum Style {
  Style1,
  Style2 }

Public Member Functions

- QwtCompassWindArrow (Style, const QColor &light=Qt::white, const QColor &dark=Qt::gray)
- virtual void draw (QPainter *, const QPoint &, int length, double direction, QPalette::ColorGroup=QPalette::Active) const

Static Public Member Functions

- static void drawStyle1Needle (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)
- static void drawStyle2Needle (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, double direction)

12.15.1 Detailed Description

An indicator for the wind direction. QwtCompassWindArrow shows the direction where the wind comes from.

- QColorGroup::Light

  Used for Style1, or the light half of Style2

- QColorGroup::Dark

  Used for the dark half of Style2

See also

QwtDial, QwtCompass

12.15.2 Member Enumeration Documentation

12.15.2.1 enum QwtCompassWindArrow::Style

Style of the arrow.

12.15.3 Constructor & Destructor Documentation

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

Constructor

Parameters

style Arrow style

light Light color

dark Dark color

12.15.4 Member Function Documentation

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

Draw the needle

Parameters

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

Implements QwtDialNeedle.

12.15.4.2 void QwtCompassWindArrow::drawStyle1Needle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, double *direction*) [static]

Draw a compass needle

Parameters

painter Painter

palette Palette

colorGroup colorGroup

center Center of the dial, start position for the needle

length Length of the needle

direction Direction of the needle, in degrees counter clockwise

12.15.4.3 void QwtCompassWindArrow::drawStyle2Needle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, double *direction*) [static]

Draw a compass needle

Parameters

painter Painter

palette Palette

colorGroup colorGroup

center Center of the dial, start position for the needle

length Length of the needle

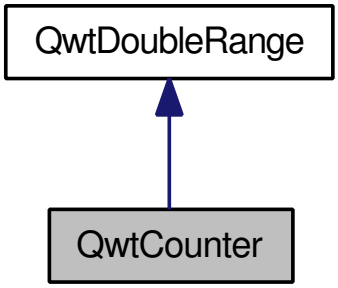
direction Direction of the needle, in degrees counter clockwise

12.16 QwtCounter Class Reference

The Counter Widget.

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

Inheritance diagram for QwtCounter:



Public Types

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

Signals

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

Public Member Functions

- QwtCounter (QWidget ∗parent=NULL)
- virtual ∼QwtCounter ()
- bool editable () const
- void setEditable (bool)
- void setNumButtons (int n)
- int numButtons () const
- void setIncSteps (QwtCounter::Button btn, int nSteps)
- int incSteps (QwtCounter::Button btn) const
- virtual void setValue (double)
- virtual QSize sizeHint () const
- virtual void polish ()
- double step () const
- void setStep (double s)
- double minVal () const
- void setMinValue (double m)
- double maxVal () const
- void setMaxValue (double m)
- void setStepButton1 (int nSteps)
- int stepButton1 () const
- void setStepButton2 (int nSteps)

- int stepButton2 () const
- void setStepButton3 (int nSteps)
- int stepButton3 () const
- virtual double value () const

Protected Member Functions

- virtual bool event (QEvent *)
- virtual void wheelEvent (QWheelEvent *)
- virtual void keyPressEvent (QKeyEvent *)
- virtual void rangeChange ()

12.16.1 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. The range can be specified using QwtDblRange::setRange(). The counter's value is an integer multiple of the step size. The number of steps by which a button increments or decrements the value can be specified using QwtCounter::setIncSteps(). The number of buttons can be changed with QwtCounter::setNumButtons().

Holding the space bar down with focus on a button is the fastest method to step through the counter values. When the counter underflows/overflows, the focus is set to the smallest up/down button and counting is disabled. Counting is re-enabled on a button release event (mouse or space bar).

Example:

```
#include "../include/qwt_counter.h>

QwtCounter *cnt;

cnt = new QwtCounter(parent, name);

cnt->setRange(0.0, 100.0, 1.0);             // From 0.0 to 100, step 1.0
cnt->setNumButtons(2);                      // Two buttons each side
cnt->setIncSteps(QwtCounter::Button1, 1);   // Button 1 increments 1 step
cnt->setIncSteps(QwtCounter::Button2, 20);  // Button 2 increments 20 steps

connect(cnt, SIGNAL(valueChanged(double)), my_class, SLOT(newValue(double)));
```

12.16.2 Member Enumeration Documentation

12.16.2.1 enum QwtCounter::Button

Button index

12.16.3 Constructor & Destructor Documentation

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

The default number of buttons is set to 2. The default increments are:

- Button 1: 1 step

- Button 2: 10 steps
- Button 3: 100 steps

Parameters

parent

12.16.3.2 QwtCounter::~QwtCounter () [virtual]

Destructor.

12.16.4 Member Function Documentation

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

This signal is emitted when a button has been released

Parameters

value The new value

12.16.4.2 bool QwtCounter::editable () const

returns whether the line edit is edatble. (default is yes)

12.16.4.3 bool QwtCounter::event (QEvent * *e*) [protected, virtual]

Handle PolishRequest events

12.16.4.4 int QwtCounter::incSteps (QwtCounter::Button *btn*) const

Returns

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

Parameters

btn One of `QwtCounter::Button1`, `QwtCounter::Button2`, `QwtCounter::Button3`

12.16.4.5 void QwtCounter::keyPressEvent (QKeyEvent * *e*) [protected, virtual]

Handle key events

- Ctrl + Qt::Key_Home Step to minValue()

- Ctrl + Qt::Key_End Step to maxValue()
- Qt::Key_Up Increment by incSteps(QwtCounter::Button1)
- Qt::Key_Down Decrement by incSteps(QwtCounter::Button1)
- Qt::Key_PageUp Increment by incSteps(QwtCounter::Button2)
- Qt::Key_PageDown Decrement by incSteps(QwtCounter::Button2)
- Shift + Qt::Key_PageUp Increment by incSteps(QwtCounter::Button3)
- Shift + Qt::Key_PageDown Decrement by incSteps(QwtCounter::Button3)

12.16.4.6 double QwtCounter::maxVal () const

returns the maximum value of the range

12.16.4.7 double QwtCounter::minVal () const

returns the minimum value of the range

12.16.4.8 int QwtCounter::numButtons () const

Returns

The number of buttons on each side of the widget.

12.16.4.9 void QwtCounter::polish () [virtual]

Sets the minimum width for the buttons

12.16.4.10 void QwtCounter::rangeChange () [protected, virtual]

Notify change of range.

This function updates the enabled property of all buttons contained in QwtCounter.

Reimplemented from QwtDoubleRange.

12.16.4.11 void QwtCounter::setEditable (bool *editable*)

Allow/disallow the user to manually edit the value.

Parameters

editable true enables editing

See also

editable()

12.16.4.12 void QwtCounter::setIncSteps (QwtCounter::Button *btn*, int *nSteps*)

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

Parameters

btn One of `QwtCounter::Button1`, `QwtCounter::Button2`, `QwtCounter::Button3`

nSteps Number of steps

12.16.4.13 void QwtCounter::setMaxValue (double *value*)

Set the maximum value of the range

Parameters

value Maximum value

See also

setMinValue(), maxVal()

12.16.4.14 void QwtCounter::setMinValue (double *value*)

Set the minimum value of the range

Parameters

value Minimum value

See also

setMaxValue(), minVal()

12.16.4.15 void QwtCounter::setNumButtons (int *n*)

Specify the number of buttons on each side of the label.

Parameters

n Number of buttons

12.16.4.16 void QwtCounter::setStep (double *stepSize*)

Set the step size

Parameters

stepSize Step size

See also

QwtDoubleRange::setStep()

Reimplemented from QwtDoubleRange.

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

Set the number of increment steps for button 1

Parameters

nSteps Number of steps

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

Set the number of increment steps for button 2

Parameters

nSteps Number of steps

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

Set the number of increment steps for button 3

Parameters

nSteps Number of steps

12.16.4.20 void QwtCounter::setValue (double *v*) `[virtual]`

Set a new value.

Parameters

v new value Calls QwtDoubleRange::setValue and does all visual updates.

See also

QwtDoubleRange::setValue()

Reimplemented from QwtDoubleRange.

12.16.4.21 QSize QwtCounter::sizeHint () const [virtual]

A size hint.

12.16.4.22 double QwtCounter::step () const

returns the step size

Reimplemented from QwtDoubleRange.

12.16.4.23 int QwtCounter::stepButton1 () const

returns the number of increment steps for button 1

12.16.4.24 int QwtCounter::stepButton2 () const

returns the number of increment steps for button 2

12.16.4.25 int QwtCounter::stepButton3 () const

returns the number of increment steps for button 3

12.16.4.26 double QwtCounter::value () const [virtual]

Returns

Current value

Reimplemented from QwtDoubleRange.

12.16.4.27 void QwtCounter::valueChanged (double *value*) [signal]

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

Parameters

value The new value

12.16.4.28 void QwtCounter::wheelEvent (QWheelEvent * *e*) [protected, virtual]

Handle wheel events

Parameters

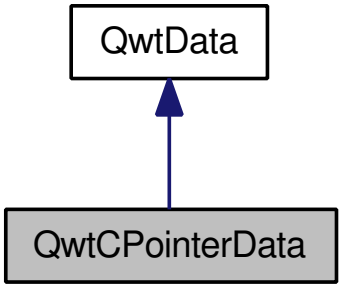
e Wheel event

12.17 QwtCPointerData Class Reference

Data class containing two pointers to memory blocks of doubles.

`#include <qwt_data.h>`

Inheritance diagram for QwtCPointerData:



Public Member Functions

- QwtCPointerData (const double *x, const double *y, size_t size)
- QwtCPointerData & operator= (const QwtCPointerData &)
- virtual QwtData * copy () const
- virtual size_t size () const
- virtual double x (size_t i) const
- virtual double y (size_t i) const
- const double * xData () const
- const double * yData () const
- virtual QwtDoubleRect boundingRect () const

12.17.1 Detailed Description

Data class containing two pointers to memory blocks of doubles.

12.17.2 Constructor & Destructor Documentation

12.17.2.1 QwtCPointerData::QwtCPointerData (const double * *x*, const double * *y*, size_t *size*)

Constructor

Parameters

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::setRawData()

12.17.3 Member Function Documentation

12.17.3.1 QwtDoubleRect QwtCPointerData::boundingRect () const [virtual]

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

Reimplemented from QwtData.

12.17.3.2 QwtData * QwtCPointerData::copy () const [virtual]

Returns

Pointer to a copy (virtual copy constructor)

Implements QwtData.

12.17.3.3 QwtCPointerData & QwtCPointerData::operator= (const QwtCPointerData & *data*)

Assignment.

12.17.3.4 size_t QwtCPointerData::size () const [virtual]

Returns

Size of the data set

Implements QwtData.

12.17.3.5 double QwtCPointerData::x (size_t *i*) const [virtual]

Return the x value of data point i

Parameters

i Index

Returns

x X value of data point i

Implements QwtData.

12.17.3.6 const double * QwtCPointerData::xData () const

Returns

Array of the x-values

12.17.3.7 double QwtCPointerData::y (size_t *i*) const [virtual]

Return the y value of data point i

Parameters

i Index

Returns

y Y value of data point i

Implements QwtData.

12.17.3.8 const double * QwtCPointerData::yData () const

Returns

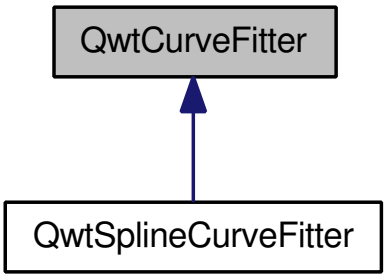
Array of the y-values

12.18 QwtCurveFitter Class Reference

Abstract base class for a curve fitter.

`#include <qwt_curve_fitter.h>`

Inheritance diagram for QwtCurveFitter:



Public Member Functions

- virtual ~QwtCurveFitter ()
- virtual QPolygonF fitCurve (const QPolygonF &polygon) const =0

Protected Member Functions

- QwtCurveFitter ()

12.18.1 Detailed Description

Abstract base class for a curve fitter.

12.18.2 Constructor & Destructor Documentation

12.18.2.1 QwtCurveFitter::~QwtCurveFitter () [virtual]

Destructor.

12.18.2.2 QwtCurveFitter::QwtCurveFitter () [protected]

Constructor.

12.18.3 Member Function Documentation

12.18.3.1 virtual QPolygonF QwtCurveFitter::fitCurve (const QPolygonF & *polygon*) const [pure virtual]

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

Parameters

polygon Series of data points

Returns

Curve points

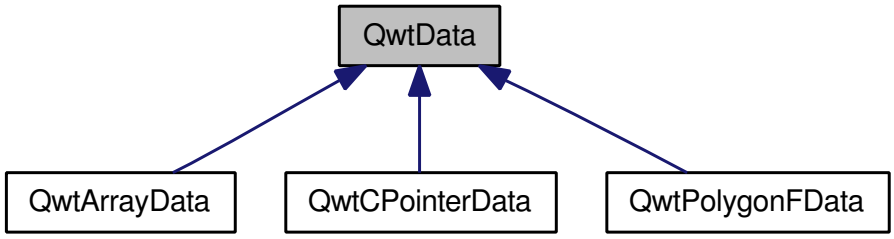
Implemented in QwtSplineCurveFitter.

12.19 QwtData Class Reference

QwtData defines an interface to any type of curve data.

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

Inheritance diagram for QwtData:



Public Member Functions

- QwtData ()
- virtual ~QwtData ()
- virtual QwtData * copy () const =0
- virtual size_t size () const =0
- virtual double x (size_t i) const =0
- virtual double y (size_t i) const =0
- virtual QwtDoubleRect boundingRect () const

Protected Member Functions

- QwtData & operator= (const QwtData &)

12.19.1 Detailed Description

QwtData defines an interface to any type of curve data. Classes, derived from QwtData may:

- store the data in almost any type of container
- calculate the data on the fly instead of storing it

12.19.2 Constructor & Destructor Documentation

12.19.2.1 QwtData::QwtData ()

Constructor.

12.19.2.2 QwtData::~QwtData () [virtual]

Destructor.

12.19.3 Member Function Documentation

12.19.3.1 QwtDoubleRect QwtData::boundingRect () const [virtual]

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

Warning

This is an slow implementation iterating over all points. It is intended to be overloaded by derived classes. In case of auto scaling boundingRect() is called for every replot, so it might be worth to implement a cache, or use x(0), x(size() - 1) for ordered data ...

Reimplemented in QwtArrayData, and QwtCPointerData.

12.19.3.2 virtual QwtData* QwtData::copy () const [pure virtual]

Returns

Pointer to a copy (virtual copy constructor)

Implemented in QwtPolygonFData, QwtArrayData, and QwtCPointerData.

12.19.3.3 QwtData& QwtData::operator= (const QwtData &) [protected]

Assignment operator (virtualized)

12.19.3.4 virtual size_t QwtData::size () const [pure virtual]

Returns

Size of the data set

Implemented in QwtPolygonFData, QwtArrayData, and QwtCPointerData.

12.19.3.5 virtual double QwtData::x (size_t *i*) const [pure virtual]

Return the x value of data point i

Parameters

i Index

Returns

x X value of data point i

Implemented in QwtPolygonFData, QwtArrayData, and QwtCPointerData.

12.19.3.6 virtual double QwtData::y (size_t *i*) const [pure virtual]

Return the y value of data point i

Parameters

i Index

Returns

y Y value of data point i

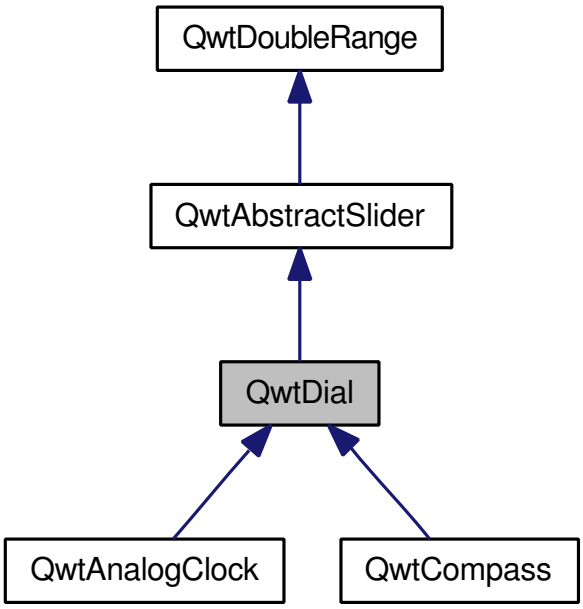
Implemented in QwtPolygonFData, QwtArrayData, and QwtCPointerData.

12.20 QwtDial Class Reference

QwtDial class provides a rounded range control.

`#include <qwt_dial.h>`

Inheritance diagram for QwtDial:



Public Types

- enum Shadow {
  Plain = QFrame::Plain,
  Raised = QFrame::Raised,
  Sunken = QFrame::Sunken }
- enum ScaleOptions {
  ScaleBackbone = 1,
  ScaleTicks = 2,
  ScaleLabel = 4 }

- enum Mode {
  RotateNeedle,
  RotateScale }
- enum Direction {
  Clockwise,
  CounterClockwise }

Public Member Functions

- QwtDial (QWidget *parent=NULL)
- virtual ~QwtDial ()
- void setFrameShadow (Shadow)
- Shadow frameShadow () const
- bool hasVisibleBackground () const
- void showBackground (bool)
- void setLineWidth (int)
- int lineWidth () const
- void setMode (Mode)
- Mode mode () const
- virtual void setWrapping (bool)
- bool wrapping () const
- virtual void setScale (int maxMajIntv, int maxMinIntv, double step=0.0)
- void setScaleArc (double min, double max)
- void setScaleOptions (int)
- void setScaleTicks (int minLen, int medLen, int majLen, int penWidth=1)
- double minScaleArc () const
- double maxScaleArc () const
- virtual void setOrigin (double)
- double origin () const
- void setDirection (Direction)
- Direction direction () const
- virtual void setNeedle (QwtDialNeedle *)
- const QwtDialNeedle * needle () const
- QwtDialNeedle * needle ()
- QRect boundingRect () const
- QRect contentsRect () const
- virtual QRect scaleContentsRect () const
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- virtual void setScaleDraw (QwtDialScaleDraw *)
- QwtDialScaleDraw * scaleDraw ()
- const QwtDialScaleDraw * scaleDraw () const

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗)
- virtual void resizeEvent (QResizeEvent ∗)
- virtual void keyPressEvent (QKeyEvent ∗)
- virtual void updateMask ()
- virtual void drawFrame (QPainter ∗p)
- virtual void drawContents (QPainter ∗) const
- virtual void drawFocusIndicator (QPainter ∗) const
- virtual void drawScale (QPainter ∗, const QPoint &center, int radius, double origin, double arcMin, double arcMax) const
- virtual void drawScaleContents (QPainter ∗painter, const QPoint &center, int radius) const
- virtual void drawNeedle (QPainter ∗, const QPoint &, int radius, double direction, QPalette::ColorGroup) const
- virtual QwtText scaleLabel (double) const
- void updateScale ()
- virtual void rangeChange ()
- virtual void valueChange ()
- virtual double getValue (const QPoint &)
- virtual void getScrollMode (const QPoint &, int &scrollMode, int &direction)

Friends

- class QwtDialScaleDraw

12.20.1 Detailed Description

QwtDial class provides a rounded range control. QwtDial is intended as base class for dial widgets like speedometers, compass widgets, clocks ...

A dial contains a scale and a needle indicating the current value of the dial. Depending on Mode one of them is fixed and the other is rotating. If not isReadOnly() the dial can be rotated by dragging the mouse or using keyboard inputs (see keyPressEvent()). A dial might be wrapping, what means a rotation below/above one limit continues on the other limit (f.e compass). The scale might cover any arc of the dial, its values are related to the origin() of the dial.

Qwt is missing a set of good looking needles (QwtDialNeedle). Contributions are very welcome.

See also

QwtCompass, QwtAnalogClock, QwtDialNeedle

Note

The examples/dials example shows different types of dials.

12.20.2 Member Enumeration Documentation

12.20.2.1 enum QwtDial::Direction

Direction of the dial

12.20.2.2 enum QwtDial::Mode

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

12.20.2.3 enum QwtDial::ScaleOptions

see QwtDial::setScaleOptions

12.20.2.4 enum QwtDial::Shadow

Frame shadow.

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

12.20.3 Constructor & Destructor Documentation

12.20.3.1 QwtDial::QwtDial (QWidget * *parent* = NULL) [explicit]

Constructor.

Parameters

parent Parent widget

Create a dial widget with no scale and no needle. The default origin is 90.0 with no valid value. It accepts mouse and keyboard inputs and has no step size. The default mode is QwtDial::RotateNeedle.

12.20.3.2 QwtDial::~QwtDial () [virtual]

Destructor.

12.20.4 Member Function Documentation

12.20.4.1 QRect QwtDial::boundingRect () const

Returns

bounding rect of the dial including the frame

See also

setLineWidth(), scaleContentsRect(), contentsRect()

12.20.4.2 QRect QwtDial::contentsRect () const

Returns

bounding rect of the circle inside the frame

See also

setLineWidth(), scaleContentsRect(), boundingRect()

12.20.4.3 QwtDial::Direction QwtDial::direction () const

Returns

Direction of the dial

The default direction of a dial is QwtDial::Clockwise

See also

setDirection()

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

Draw the contents inside the frame.

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

Parameters

painter Painter

See also

boundingRect(), contentsRect(), scaleContentsRect(), QWidget::setPalette()

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

Draw a dotted round circle, if !isReadOnly()

Parameters

painter Painter

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

Draw the frame around the dial

Parameters

painter Painter

See also

lineWidth(), frameShadow()

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

Draw the needle

Parameters

painter Painter

center Center of the dial

radius Length for the needle

direction Direction of the needle in degrees, counter clockwise

cg ColorGroup

Reimplemented in QwtAnalogClock.

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

Draw the scale

Parameters

painter Painter

center Center of the dial

radius Radius of the scale

origin Origin of the scale

minArc Minimum of the arc

maxArc Minimum of the arc

See also

QwtAbstractScaleDraw::setAngleRange()

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

Draw the contents inside the scale

Paints nothing.

Parameters

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

Reimplemented in QwtCompass.

12.20.4.10 QwtDial::Shadow QwtDial::frameShadow () const

Returns

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

12.20.4.11 void QwtDial::getScrollMode (const QPoint & *pos*, int & *scrollMode*, int & *direction*) [protected, virtual]

See QwtAbstractSlider::getScrollMode()

Parameters

pos point where the mouse was pressed

Return values

scrollMode The scrolling mode
direction direction: 1, 0, or -1.

See also

QwtAbstractSlider::getScrollMode()

Implements QwtAbstractSlider.

12.20.4.12 double QwtDial::getValue (const QPoint & *pos*) [protected, virtual]

Find the value for a given position

Parameters

pos Position

Returns

Value

Implements QwtAbstractSlider.

12.20.4.13 bool QwtDial::hasVisibleBackground () const

true when the area outside of the frame is visible

See also

showBackground(), setMask()

12.20.4.14 void QwtDial::keyPressEvent (QKeyEvent * *event*) [protected, virtual]

Handles key events

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

Parameters

event Key event

See also

isReadOnly()

Reimplemented from QwtAbstractSlider.

Reimplemented in QwtCompass.

12.20.4.15 int QwtDial::lineWidth () const

Returns

Line width of the frame

See also

setLineWidth(), frameShadow(), lineWidth()

12.20.4.16 double QwtDial::maxScaleArc () const

Returns

Upper limit of the scale arc

12.20.4.17 QSize QwtDial::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

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

12.20.4.18 double QwtDial::minScaleArc () const

Returns

Lower limit of the scale arc

12.20.4.19 QwtDial::Mode QwtDial::mode () const

Returns

mode of the dial.

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

The default mode is QwtDial::RotateNeedle.

See also

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

12.20.4.20 QwtDialNeedle * QwtDial::needle ()

Returns

needle

See also

setNeedle()

12.20.4.21 const QwtDialNeedle * QwtDial::needle () const

Returns

needle

See also

setNeedle()

12.20.4.22 double QwtDial::origin () const

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

Returns

Origin of the dial

See also

setOrigin()

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

Paint the dial

Parameters

e Paint event

12.20.4.24 void QwtDial::rangeChange () [protected, virtual]

QwtDoubleRange update hook.

Reimplemented from QwtDoubleRange.

12.20.4.25 void QwtDial::resizeEvent (QResizeEvent * *e*) [protected, virtual]

Resize the dial widget

Parameters

e Resize event

12.20.4.26 QRect QwtDial::scaleContentsRect () const [virtual]

Returns

rect inside the scale

See also

setLineWidth(), boundingRect(), contentsRect()

12.20.4.27 const QwtDialScaleDraw * QwtDial::scaleDraw () const

Return the scale draw.

12.20.4.28 QwtDialScaleDraw * QwtDial::scaleDraw ()

Return the scale draw.

12.20.4.29 QwtText QwtDial::scaleLabel (double *value*) const [protected, virtual]

Find the label for a value

Parameters

value Value

Returns

label

Reimplemented in QwtAnalogClock, and QwtCompass.

12.20.4.30 void QwtDial::setDirection (Direction *direction*)

Set the direction of the dial (clockwise/counterclockwise)

Direction direction

See also

direction()

12.20.4.31 void QwtDial::setFrameShadow (Shadow *shadow*)

Sets the frame shadow value from the frame style.

Parameters

shadow Frame shadow

See also

setLineWidth(), QFrame::setFrameShadow()

12.20.4.32 void QwtDial::setLineWidth (int *lineWidth*)

Sets the line width

Parameters

lineWidth Line width

See also

setFrameShadow()

12.20.4.33 void QwtDial::setMode (Mode *mode*)

Change the mode of the meter.

Parameters

mode New mode

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

The default mode is QwtDial::RotateNeedle.

See also

mode(), setValue(), setOrigin()

12.20.4.34 void QwtDial::setNeedle (QwtDialNeedle * *needle*) [virtual]

Set a needle for the dial

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

Parameters

needle Needle

Warning

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

12.20.4.35 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()

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

Change the intervals of the scale

See also

QwtAbstractScaleDraw::setScale()

12.20.4.37 void QwtDial::setScaleArc (double *minArc*, double *maxArc*)

Change the arc of the scale

Parameters

minArc Lower limit

maxArc Upper limit

12.20.4.38 void QwtDial::setScaleDraw (QwtDialScaleDraw * *scaleDraw*) [virtual]

Set an individual scale draw

Parameters

scaleDraw Scale draw

Warning

The previous scale draw is deleted

12.20.4.39 void QwtDial::setScaleOptions (int *options*)

A wrapper method for accessing the scale draw.

- options == 0
  No visible scale: setScaleDraw(NULL)

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

See also

QwtAbstractScaleDraw::enableComponent()

12.20.4.40 void QwtDial::setScaleTicks (int *minLen*, int *medLen*, int *majLen*, int *penWidth* = 1)

Assign length and width of the ticks

Parameters

minLen Length of the minor ticks
medLen Length of the medium ticks
majLen Length of the major ticks
penWidth Width of the pen for all ticks

See also

QwtAbstractScaleDraw::setTickLength(), QwtDialScaleDraw::setPenWidth()

12.20.4.41 void QwtDial::setWrapping (bool *wrapping*) [virtual]

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

Parameters

wrapping en/disables wrapping

See also

wrapping(), QwtDoubleRange::periodic()

Note

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

12.20.4.42 void QwtDial::showBackground (bool *show*)

Show/Hide the area outside of the frame

Parameters

show Show if true, hide if false

See also

hasVisibleBackground(), setMask()

Warning

When QwtDial is a toplevel widget the window border might disappear too.

12.20.4.43 QSize QwtDial::sizeHint () const [virtual]

Returns

Size hint

12.20.4.44 void QwtDial::updateMask () [protected, virtual]

Update the mask of the dial.

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

See also

showBackground(), hasVisibleBackground()

12.20.4.45 void QwtDial::updateScale () [protected]

Update the scale with the current attributes

See also

setScale()

12.20.4.46 void QwtDial::valueChange () [protected, virtual]

QwtDoubleRange update hook.

Reimplemented from QwtAbstractSlider.

12.20.4.47 bool QwtDial::wrapping () const

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

See also

setWrapping(), QwtDoubleRange::setPeriodic()

Note

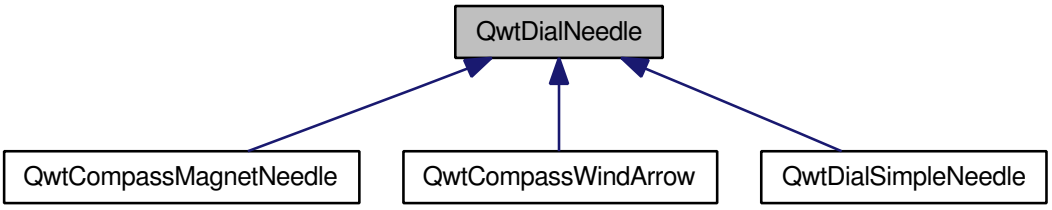
The meaning of wrapping is like the wrapping property of QSpinBox, but not like it is used in QDial.

12.21 QwtDialNeedle Class Reference

Base class for needles that can be used in a QwtDial.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtDialNeedle:



Public Member Functions

- QwtDialNeedle ()
- virtual ~QwtDialNeedle ()
- virtual void draw (QPainter *painter, const QPoint &center, int length, double direction, QPalette::ColorGroup cg=QPalette::Active) const =0
- virtual void setPalette (const QPalette &)
- const QPalette & palette () const

Static Protected Member Functions

- static void drawKnob (QPainter *, const QPoint &pos, int width, const QBrush &, bool sunken)

12.21.1 Detailed Description

Base class for needles that can be used in a QwtDial. QwtDialNeedle is a pointer that indicates a value by pointing to a specific direction.

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

See also

QwtDial, QwtCompass

12.21.2 Constructor & Destructor Documentation

12.21.2.1 QwtDialNeedle::QwtDialNeedle ()

Constructor.

12.21.2.2 QwtDialNeedle::~QwtDialNeedle () [virtual]

Destructor.

12.21.3 Member Function Documentation

12.21.3.1 virtual void QwtDialNeedle::draw (QPainter * *painter*, const QPoint & *center*, int *length*, double *direction*, QPalette::ColorGroup *cg* = `QPalette::Active`) const `[pure virtual]`

Draw the needle

Parameters

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

cg Color group, used for painting

Implemented in QwtDialSimpleNeedle, QwtCompassMagnetNeedle, and QwtCompassWindArrow.

12.21.3.2 void QwtDialNeedle::drawKnob (QPainter * *painter*, const QPoint & *pos*, int *width*, const QBrush & *brush*, bool *sunken*) `[static, protected]`

Draw the knob.

12.21.3.3 const QPalette & QwtDialNeedle::palette () const

Returns

the palette of the needle.

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

Sets the palette for the needle.

Parameters

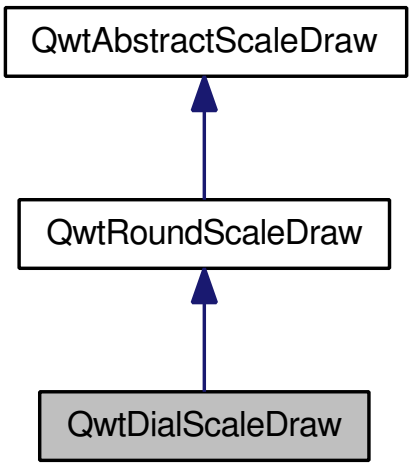
palette New Palette

12.22 QwtDialScaleDraw Class Reference

A special scale draw made for QwtDial.

`#include <qwt_dial.h>`

Inheritance diagram for QwtDialScaleDraw:



Public Member Functions

- QwtDialScaleDraw (QwtDial ∗)
- virtual QwtText label (double value) const
- void setPenWidth (uint)
- uint penWidth () const

12.22.1 Detailed Description

A special scale draw made for QwtDial.

See also

QwtDial, QwtCompass

12.22.2 Constructor & Destructor Documentation

12.22.2.1 QwtDialScaleDraw::QwtDialScaleDraw (QwtDial ∗ *parent*) `[explicit]`

Constructor

Parameters

parent Parent dial widget

12.22.3 Member Function Documentation

12.22.3.1 QwtText QwtDialScaleDraw::label (double *value*) const `[virtual]`

Call QwtDial::scaleLabel of the parent dial widget.

Parameters

value Value to display

See also

QwtDial::scaleLabel()

Reimplemented from QwtAbstractScaleDraw.

12.22.3.2 uint QwtDialScaleDraw::penWidth () const

Returns

Pen width used for painting the scale

See also

setPenWidth, QwtDial::drawScale()

12.22.3.3 void QwtDialScaleDraw::setPenWidth (uint *penWidth*)

Set the pen width used for painting the scale

Parameters

penWidth Pen width

See also

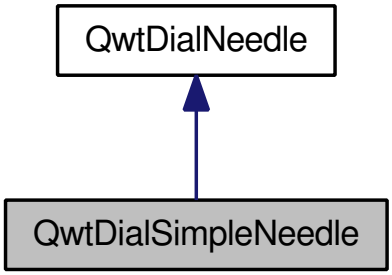
penWidth(), QwtDial::drawScale()

12.23 QwtDialSimpleNeedle Class Reference

A needle for dial widgets.

`#include <qwt_dial_needle.h>`

Inheritance diagram for QwtDialSimpleNeedle:



Public Types

- enum Style {
  Arrow,
  Ray }

Public Member Functions

- QwtDialSimpleNeedle (Style, bool hasKnob=true, const QColor &mid=Qt::gray, const QColor &base=Qt::darkGray)
- virtual void draw (QPainter *, const QPoint &, int length, double direction, QPalette::ColorGroup=QPalette::Active) const
- void setWidth (int width)
- int width () const

Static Public Member Functions

- static void drawArrowNeedle (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, int width, double direction, bool hasKnob)
- static void drawRayNeedle (QPainter *, const QPalette &, QPalette::ColorGroup, const QPoint &, int length, int width, double direction, bool hasKnob)

12.23.1 Detailed Description

A needle for dial widgets. The following colors are used:

- QColorGroup::Mid

  Pointer
- QColorGroup::base

  Knob

See also

QwtDial, QwtCompass

12.23.2 Member Enumeration Documentation

12.23.2.1 enum QwtDialSimpleNeedle::Style

Style of the needle.

12.23.3 Constructor & Destructor Documentation

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

Constructor

Parameters

style Style

hasKnob With/Without knob

mid Middle color

base Base color

12.23.4 Member Function Documentation

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

Draw the needle

Parameters

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

Implements QwtDialNeedle.

12.23.4.2 void QwtDialSimpleNeedle::drawArrowNeedle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, int *width*, double *direction*, bool *hasKnob*) `[static]`

Draw a needle looking like an arrow

Parameters

painter Painter
palette Palette
colorGroup Color group
center center of the needle
length Length of the needle
width Width of the needle
direction Current Direction
hasKnob With/Without knob

12.23.4.3 void QwtDialSimpleNeedle::drawRayNeedle (QPainter * *painter*, const QPalette & *palette*, QPalette::ColorGroup *colorGroup*, const QPoint & *center*, int *length*, int *width*, double *direction*, bool *hasKnob*) `[static]`

Draw a needle looking like a ray

Parameters

painter Painter
palette Palette
colorGroup Color group
center center of the needle
length Length of the needle
width Width of the needle
direction Current Direction
hasKnob With/Without knob

12.23.4.4 void QwtDialSimpleNeedle::setWidth (int *width*)

Set the width of the needle

Parameters

width Width

See also

width()

12.23.4.5 int QwtDialSimpleNeedle::width () const

Returns

the width of the needle

See also

setWidth()

12.24 QwtDoubleInterval Class Reference

A class representing an interval.

`#include <qwt_double_interval.h>`

Public Types

- enum BorderMode {
  IncludeBorders = 0,
  ExcludeMinimum = 1,
  ExcludeMaximum = 2,
  ExcludeBorders = ExcludeMinimum | ExcludeMaximum }

Public Member Functions

- QwtDoubleInterval ()
- QwtDoubleInterval (double minValue, double maxValue, int borderFlags=IncludeBorders)
- void setInterval (double minValue, double maxValue, int borderFlags=IncludeBorders)
- QwtDoubleInterval normalized () const
- QwtDoubleInterval inverted () const
- QwtDoubleInterval limited (double minValue, double maxValue) const
- int operator== (const QwtDoubleInterval &) const
- int operator!= (const QwtDoubleInterval &) const
- void setBorderFlags (int)
- int borderFlags () const
- double minValue () const

- double maxValue () const
- double width () const
- void setMinValue (double)
- void setMaxValue (double)
- bool contains (double value) const
- bool intersects (const QwtDoubleInterval &) const
- QwtDoubleInterval intersect (const QwtDoubleInterval &) const
- QwtDoubleInterval unite (const QwtDoubleInterval &) const
- QwtDoubleInterval operator| (const QwtDoubleInterval &) const
- QwtDoubleInterval operator& (const QwtDoubleInterval &) const
- QwtDoubleInterval & operator|= (const QwtDoubleInterval &)
- QwtDoubleInterval & operator&= (const QwtDoubleInterval &)
- QwtDoubleInterval extend (double value) const
- QwtDoubleInterval operator| (double) const
- QwtDoubleInterval & **operator|=** (double)
- bool isValid () const
- bool isNull () const
- void invalidate ()
- QwtDoubleInterval symmetrize (double value) const

12.24.1 Detailed Description

A class representing an interval. The interval is represented by 2 doubles, the lower and the upper limit.

12.24.2 Member Enumeration Documentation

12.24.2.1 enum QwtDoubleInterval::BorderMode

Flag indicating if a border is included/excluded from an interval

- IncludeBorders
  min/max values are inside the interval
- ExcludeMinimum
  min value is not included in the interval
- ExcludeMaximum
  max value is not included in the interval
- ExcludeBorders
  min/max values are not included in the interval

See also

setBorderMode(), testBorderMode()

12.24.3 Constructor & Destructor Documentation

12.24.3.1 QwtDoubleInterval::QwtDoubleInterval () [inline]

Default Constructor.

Creates an invalid interval [0.0, -1.0]

See also

setInterval(), isValid()

12.24.3.2 QwtDoubleInterval::QwtDoubleInterval (double *minValue*, double *maxValue*, int *borderFlags* = IncludeBorders) [inline]

Constructor

Build an interval with from min/max values

Parameters

minValue Minimum value

maxValue Maximum value

borderFlags Include/Exclude borders

12.24.4 Member Function Documentation

12.24.4.1 int QwtDoubleInterval::borderFlags () const [inline]

Returns

Border flags

See also

setBorderFlags()

12.24.4.2 bool QwtDoubleInterval::contains (double *value*) const

Test if a value is inside an interval

Parameters

value Value

Returns

true, if value >= minValue() && value <= maxValue()

12.24.4.3 QwtDoubleInterval QwtDoubleInterval::extend (double *value*) const

Extend the interval

If value is below minValue, value becomes the lower limit. If value is above maxValue, value becomes the upper limit.

extend has no effect for invalid intervals

Parameters

value Value

See also

isValid()

12.24.4.4 QwtDoubleInterval QwtDoubleInterval::intersect (const QwtDoubleInterval & *other*) const

Intersect 2 intervals.

12.24.4.5 bool QwtDoubleInterval::intersects (const QwtDoubleInterval & *other*) const

Test if two intervals overlap

12.24.4.6 void QwtDoubleInterval::invalidate () `[inline]`

Invalidate the interval

The limits are set to interval [0.0, -1.0]

See also

isValid()

12.24.4.7 QwtDoubleInterval QwtDoubleInterval::inverted () const

Invert the limits of the interval

Returns

Inverted interval

See also

normalized()

12.24.4.8 bool QwtDoubleInterval::isNull () const [inline]

Returns

true, if isValid() && (minValue() >= maxValue())

12.24.4.9 bool QwtDoubleInterval::isValid () const [inline]

A interval is valid when minValue() <= maxValue(). In case of QwtDoubleInterval::ExcludeBorders it is true when minValue() < maxValue()

12.24.4.10 QwtDoubleInterval QwtDoubleInterval::limited (double *lowerBound*, double *upperBound*) const

Limit the interval, keeping the border modes

Parameters

lowerBound Lower limit

upperBound Upper limit

Returns

Limited interval

12.24.4.11 double QwtDoubleInterval::maxValue () const [inline]

Returns

Upper limit of the interval

12.24.4.12 double QwtDoubleInterval::minValue () const [inline]

Returns

Lower limit of the interval

12.24.4.13 QwtDoubleInterval QwtDoubleInterval::normalized () const

Normalize the limits of the interval.

If maxValue() < minValue() the limits will be inverted.

Returns

Normalized interval

See also

isValid(), inverted()

12.24.4.14 int QwtDoubleInterval::operator!= (const QwtDoubleInterval & *other*) const [inline]

Compare two intervals.

12.24.4.15 QwtDoubleInterval QwtDoubleInterval::operator& (const QwtDoubleInterval & *interval*) const [inline]

Intersection of two intervals

See also

intersect()

12.24.4.16 QwtDoubleInterval & QwtDoubleInterval::operator&= (const QwtDoubleInterval & *interval*)

Intersects this interval with the given interval.

12.24.4.17 int QwtDoubleInterval::operator== (const QwtDoubleInterval & *other*) const [inline]

Compare two intervals.

12.24.4.18 QwtDoubleInterval QwtDoubleInterval::operator| (double *value*) const [inline]

Extend an interval

See also

extend()

12.24.4.19 QwtDoubleInterval QwtDoubleInterval::operator| (const QwtDoubleInterval & *interval*) const [inline]

Union of two intervals

See also

unite()

12.24.4.20 QwtDoubleInterval & QwtDoubleInterval::operator|= (const QwtDoubleInterval & *interval*)

Unites this interval with the given interval.

12.24.4.21 void QwtDoubleInterval::setBorderFlags (int *borderFlags*) [inline]

Change the border flags

Parameters

borderFlags Or'd BorderMode flags

See also

borderFlags()

12.24.4.22 void QwtDoubleInterval::setInterval (double *minValue*, double *maxValue*, int *borderFlags* = IncludeBorders) [inline]

Assign the limits of the interval

Parameters

minValue Minimum value

maxValue Maximum value

borderFlags Include/Exclude borders

12.24.4.23 void QwtDoubleInterval::setMaxValue (double *maxValue*) [inline]

Assign the upper limit of the interval

Parameters

maxValue Maximum value

12.24.4.24 void QwtDoubleInterval::setMinValue (double *minValue*) [inline]

Assign the lower limit of the interval

Parameters

minValue Minimum value

12.24.4.25 QwtDoubleInterval QwtDoubleInterval::symmetrize (double *value*) const

Adjust the limit that is closer to value, so that value becomes the center of the interval.

Parameters

value Center

Returns

Interval with value as center

12.24.4.26 QwtDoubleInterval QwtDoubleInterval::unite (const QwtDoubleInterval & *other*) const

Unite 2 intervals.

12.24.4.27 double QwtDoubleInterval::width () const [inline]

Return the width of an interval The width of invalid intervals is 0.0, otherwise the result is maxValue() - minValue().

See also

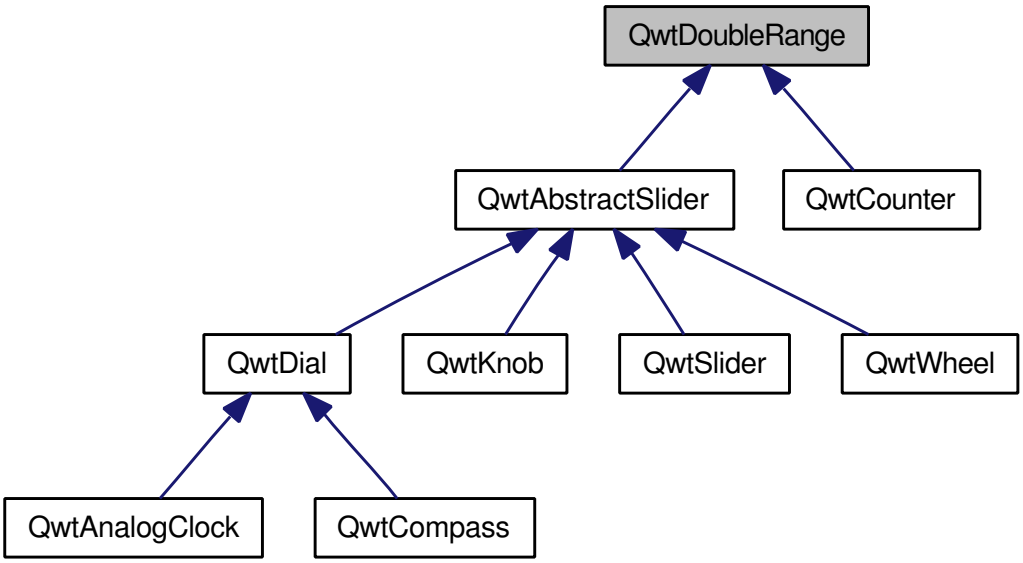
isValid()

12.25 QwtDoubleRange Class Reference

A class which controls a value within an interval.

`#include <qwt_double_range.h>`

Inheritance diagram for QwtDoubleRange:



Public Member Functions

- QwtDoubleRange ()
- virtual ~QwtDoubleRange ()
- void setRange (double vmin, double vmax, double vstep=0.0, int pagesize=1)
- void setValid (bool)
- bool isValid () const
- virtual void setValue (double)
- double value () const
- void setPeriodic (bool tf)
- bool periodic () const
- void setStep (double)
- double step () const
- double maxValue () const
- double minValue () const
- int pageSize () const
- virtual void incValue (int)
- virtual void incPages (int)
- virtual void fitValue (double)

Protected Member Functions

- double exactValue () const
- double exactPrevValue () const
- double prevValue () const
- virtual void valueChange ()
- virtual void stepChange ()
- virtual void rangeChange ()

12.25.1 Detailed Description

A class which controls a value within an interval. This class is useful as a base class or a member for sliders. It represents an interval of type double within which a value can be moved. The value can be either an arbitrary point inside the interval (see QwtDoubleRange::setValue), or it can be fitted into a step raster (see QwtDoubleRange::fitValue and QwtDoubleRange::incValue).

As a special case, a QwtDoubleRange can be periodic, which means that a value outside the interval will be mapped to a value inside the interval when QwtDoubleRange::setValue(), QwtDoubleRange::fitValue(), QwtDoubleRange::incValue() or QwtDoubleRange::incPages() are called.

12.25.2 Constructor & Destructor Documentation

12.25.2.1 QwtDoubleRange::QwtDoubleRange ()

The range is initialized to [0.0, 100.0], the step size to 1.0, and the value to 0.0.

12.25.2.2 QwtDoubleRange::~QwtDoubleRange () [virtual]

Destroys the QwtDoubleRange.

12.25.3 Member Function Documentation

12.25.3.1 double QwtDoubleRange::exactPrevValue () const [protected]

Returns the exact previous value.

12.25.3.2 double QwtDoubleRange::exactValue () const [protected]

Returns the exact value.

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

12.25.3.3 void QwtDoubleRange::fitValue (double *x*) [virtual]

Adjust the value to the closest point in the step raster.

Parameters

x value

Warning

The value is clipped when it lies outside the range. When the range is QwtDoubleRange::periodic, it will be mapped to a point in the interval such that

```
new value := x + n * (max. value - min. value)
```

with an integer number n.

Reimplemented in QwtAbstractSlider.

12.25.3.4 void QwtDoubleRange::incPages (int *nPages*) [virtual]

Increment the value by a specified number of pages.

Parameters

nPages Number of pages to increment. A negative number decrements the value.

Warning

The Page size is specified in the constructor.

12.25.3.5 void QwtDoubleRange::incValue (int *nSteps*) [virtual]

Increment the value by a specified number of steps.

Parameters

nSteps Number of steps to increment

Warning

As a result of this operation, the new value will always be adjusted to the step raster.

Reimplemented in QwtAbstractSlider.

12.25.3.6 bool QwtDoubleRange::isValid () const

Indicates if the value is valid.

Reimplemented in QwtAbstractSlider.

12.25.3.7 double QwtDoubleRange::maxValue () const

Returns the value of the second border of the range.

maxValue returns the value which has been specified as the second parameter in QwtDoubleRange::setRange.

See also

setRange()

12.25.3.8 double QwtDoubleRange::minValue () const

Returns the value at the first border of the range.

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

See also

setRange()

12.25.3.9 int QwtDoubleRange::pageSize () const

Returns the page size in steps.

12.25.3.10 bool QwtDoubleRange::periodic () const

Returns true if the range is periodic.

See also

setPeriodic()

12.25.3.11 double QwtDoubleRange::prevValue () const [protected]

Returns the previous value.

12.25.3.12 void QwtDoubleRange::rangeChange () [protected, virtual]

Notify a change of the range.

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

Reimplemented in QwtCounter, QwtDial, and QwtSlider.

12.25.3.13 void QwtDoubleRange::setPeriodic (bool *tf*)

Make the range periodic.

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

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

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

Parameters

tf true for a periodic range

12.25.3.14 void QwtDoubleRange::setRange (double *vmin*, double *vmax*, double *vstep* = 0.0, int *pageSize* = 1)

Specify range and step size.

Parameters

vmin lower boundary of the interval

vmax higher boundary of the interval

vstep step width

pageSize page size in steps

Warning

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

12.25.3.15 void QwtDoubleRange::setStep (double *vstep*)

Change the step raster.

Parameters

vstep new step width

Warning

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

Reimplemented in QwtCounter.

12.25.3.16 void QwtDoubleRange::setValid (bool *isValid*)

Set the value to be valid/invalid.

Reimplemented in QwtAbstractSlider.

12.25.3.17 void QwtDoubleRange::setValue (double *x*) `[virtual]`

Set a new value without adjusting to the step raster.

Parameters

x new value

Warning

The value is clipped when it lies outside the range. When the range is QwtDoubleRange::periodic, it will be mapped to a point in the interval such that

```
new value := x + n * (max. value - min. value)
```

with an integer number n.

Reimplemented in QwtAbstractSlider, and QwtCounter.

12.25.3.18 double QwtDoubleRange::step () const

Returns

the step size

See also

setStep(), setRange()

Reimplemented in QwtCounter.

12.25.3.19 void QwtDoubleRange::stepChange () [protected, virtual]

Notify a change of the step size.

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

12.25.3.20 double QwtDoubleRange::value () const

Returns the current value.

Reimplemented in QwtCounter.

12.25.3.21 void QwtDoubleRange::valueChange () [protected, virtual]

Notify a change of value.

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

Reimplemented in QwtAbstractSlider, QwtDial, QwtSlider, and QwtWheel.

12.26 QwtDynGridLayout Class Reference

The QwtDynGridLayout class lays out widgets in a grid, adjusting the number of columns and rows to the current size.

```
#include <qwt_dyngrid_layout.h>
```

Public Member Functions

- QwtDynGridLayout (QWidget *, int margin=0, int space=-1)
- QwtDynGridLayout (int space=-1)
- virtual ~QwtDynGridLayout ()
- virtual void invalidate ()
- void setMaxCols (uint maxCols)
- uint maxCols () const

- uint numRows () const
- uint numCols () const
- virtual void addItem (QLayoutItem ∗)
- virtual QLayoutItem ∗ itemAt (int index) const
- virtual QLayoutItem ∗ takeAt (int index)
- virtual int count () const
- void setExpandingDirections (Qt::Orientations)
- virtual Qt::Orientations expandingDirections () const
- QList< QRect > layoutItems (const QRect &, uint numCols) const
- virtual int maxItemWidth () const
- virtual void setGeometry (const QRect &rect)
- virtual bool hasHeightForWidth () const
- virtual int heightForWidth (int) const
- virtual QSize sizeHint () const
- virtual bool isEmpty () const
- uint itemCount () const
- virtual uint columnsForWidth (int width) const

Protected Member Functions

- void layoutGrid (uint numCols, QwtArray< int > &rowHeight, QwtArray< int > &colWidth) const
- void stretchGrid (const QRect &rect, uint numCols, QwtArray< int > &rowHeight, QwtArray< int > &colWidth) const

12.26.1 Detailed Description

The QwtDynGridLayout class lays out widgets in a grid, adjusting the number of columns and rows to the current size. QwtDynGridLayout takes the space it gets, divides it up into rows and columns, and puts each of the widgets it manages into the correct cell(s). It lays out as many number of columns as possible (limited by maxCols()).

12.26.2 Constructor & Destructor Documentation

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

Parameters

parent Parent widget

margin Margin

spacing Spacing

12.26.2.2 QwtDynGridLayout::QwtDynGridLayout (int *spacing* = -1) [explicit]

Parameters

spacing Spacing

12.26.2.3 QwtDynGridLayout::~QwtDynGridLayout () [virtual]

Destructor.

12.26.3 Member Function Documentation

12.26.3.1 void QwtDynGridLayout::addItem (QLayoutItem * *item*) [virtual]

Adds item to the next free position.

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

Calculate the number of columns for a given width. It tries to use as many columns as possible (limited by maxCols())

Parameters

width Available width for all columns

See also

maxCols(), setMaxCols()

12.26.3.3 int QwtDynGridLayout::count () const [virtual]

Returns

Number of items in the layout

12.26.3.4 Qt::Orientations QwtDynGridLayout::expandingDirections () const [virtual]

Returns whether this layout can make use of more space than sizeHint(). A value of Qt::Vertical or Qt::Horizontal means that it wants to grow in only one dimension, while Qt::Vertical | Qt::Horizontal means that it wants to grow in both dimensions.

See also

setExpandingDirections()

12.26.3.5 bool QwtDynGridLayout::hasHeightForWidth () const [virtual]

Returns

true: QwtDynGridLayout implements heightForWidth.

See also

heightForWidth()

12.26.3.6 int QwtDynGridLayout::heightForWidth (int *width*) const [virtual]

Returns

The preferred height for this layout, given the width w.

See also

hasHeightForWidth()

12.26.3.7 void QwtDynGridLayout::invalidate () [virtual]

Invalidate all internal caches.

12.26.3.8 bool QwtDynGridLayout::isEmpty () const [virtual]

Returns

true if this layout is empty.

12.26.3.9 QLayoutItem * QwtDynGridLayout::itemAt (int *index*) const [virtual]

Find the item at a spcific index

Parameters

index Index

See also

takeAt()

12.26.3.10 uint QwtDynGridLayout::itemCount () const

Returns

number of layout items

12.26.3.11 void QwtDynGridLayout::layoutGrid (uint *numCols*, QwtArray< int > & *rowHeight*, QwtArray< int > & *colWidth*) const [protected]

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

Parameters

numCols Number of columns.

rowHeight Array where to fill in the calculated row heights.

colWidth Array where to fill in the calculated column widths.

12.26.3.12 QList< QRect > QwtDynGridLayout::layoutItems (const QRect & *rect*, uint *numCols*) const

Calculate the geometries of the layout items for a layout with numCols columns and a given rect.

Parameters

rect Rect where to place the items

numCols Number of columns

Returns

item geometries

12.26.3.13 uint QwtDynGridLayout::maxCols () const

Return the upper limit for the number of columns. 0 means unlimited, what is the default.

See also

setMaxCols()

12.26.3.14 int QwtDynGridLayout::maxItemWidth () const [virtual]

Returns

the maximum width of all layout items

12.26.3.15 uint QwtDynGridLayout::numCols () const

Returns

Number of columns of the current layout.

See also

numRows()

Warning

The number of columns might change whenever the geometry changes

12.26.3.16 uint QwtDynGridLayout::numRows () const

Returns

Number of rows of the current layout.

See also

numCols()

Warning

The number of rows might change whenever the geometry changes

12.26.3.17 void QwtDynGridLayout::setExpandingDirections (Qt::Orientations *expanding*)

Set whether this layout can make use of more space than sizeHint(). A value of Qt::Vertical or Qt::Horizontal means that it wants to grow in only one dimension, while Qt::Vertical | Qt::Horizontal means that it wants to grow in both dimensions. The default value is 0.

Parameters

expanding Or'd orientations

See also

expandingDirections()

12.26.3.18 void QwtDynGridLayout::setGeometry (const QRect & *rect*) [virtual]

Reorganizes columns and rows and resizes managed widgets within the rectangle rect.

Parameters

rect Layout geometry

12.26.3.19 void QwtDynGridLayout::setMaxCols (uint *maxCols*)

Limit the number of columns.

Parameters

maxCols upper limit, 0 means unlimited

See also

maxCols()

12.26.3.20 QSize QwtDynGridLayout::sizeHint () const [virtual]

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

See also

maxCols(), setMaxCols()

12.26.3.21 void QwtDynGridLayout::stretchGrid (const QRect & *rect*, uint *numCols*, QwtArray< int > & *rowHeight*, QwtArray< int > & *colWidth*) const [protected]

Stretch columns in case of expanding() & QSizePolicy::Horizontal and rows in case of expanding() & QSizePolicy::Vertical to fill the entire rect. Rows and columns are stretched with the same factor.

See also

setExpanding(), expanding()

12.26.3.22 QLayoutItem * QwtDynGridLayout::takeAt (int *index*) [virtual]

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

Parameters

index Index

See also

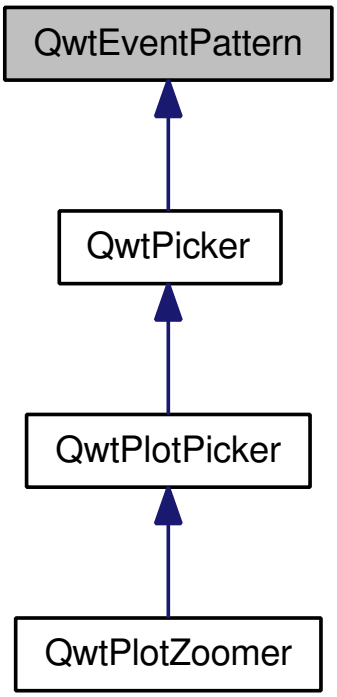
itemAt()

12.27 QwtEventPattern Class Reference

A collection of event patterns.

`#include <qwt_event_pattern.h>`

Inheritance diagram for QwtEventPattern:



Classes

- class KeyPattern
  A pattern for key events.
- class MousePattern
  A pattern for mouse events.

Public Types

- enum MousePatternCode {
  MouseSelect1,
  MouseSelect2,
  MouseSelect3,
  MouseSelect4,
  MouseSelect5,
  MouseSelect6,
  MousePatternCount }
- enum KeyPatternCode {
  KeySelect1,
  KeySelect2,
  KeyAbort,

KeyLeft,
KeyRight,
KeyUp,
KeyDown,
KeyRedo,
KeyUndo,
KeyHome,
KeyPatternCount }

Public Member Functions

- QwtEventPattern ()
- virtual ~QwtEventPattern ()
- void initMousePattern (int numButtons)
- void initKeyPattern ()
- void setMousePattern (uint pattern, int button, int state=Qt::NoButton)
- void setKeyPattern (uint pattern, int key, int state=Qt::NoButton)
- void setMousePattern (const QwtArray< MousePattern > &)
- void setKeyPattern (const QwtArray< KeyPattern > &)
- const QwtArray< MousePattern > & mousePattern () const
- const QwtArray< KeyPattern > & keyPattern () const
- QwtArray< MousePattern > & mousePattern ()
- QwtArray< KeyPattern > & keyPattern ()
- bool mouseMatch (uint pattern, const QMouseEvent *) const
- bool keyMatch (uint pattern, const QKeyEvent *) const

Protected Member Functions

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

12.27.1 Detailed Description

A collection of event patterns. QwtEventPattern introduces an level of indirection for mouse and keyboard inputs. Those are represented by symbolic names, so the application code can be configured by individual mappings.

See also

QwtPicker, QwtPickerMachine, QwtPlotZoomer

12.27.2 Member Enumeration Documentation

12.27.2.1 enum QwtEventPattern::KeyPatternCode

Symbolic keyboard input codes.

Default initialization:

- KeySelect1
  Qt::Key_Return
- KeySelect2
  Qt::Key_Space
- KeyAbort
  Qt::Key_Escape
- KeyLeft
  Qt::Key_Left
- KeyRight
  Qt::Key_Right
- KeyUp
  Qt::Key_Up
- KeyDown
  Qt::Key_Down
- KeyUndo
  Qt::Key_Minus
- KeyRedo
  Qt::Key_Plus
- KeyHome
  Qt::Key_Escape

12.27.2.2 enum QwtEventPattern::MousePatternCode

Symbolic mouse input codes.

The default initialization for 3 button mice is:

- MouseSelect1
  Qt::LeftButton
- MouseSelect2
  Qt::RightButton
- MouseSelect3
  Qt::MidButton
- MouseSelect4
  Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
  Qt::RightButton + Qt::ShiftButton

- MouseSelect6
  Qt::MidButton + Qt::ShiftButton

The default initialization for 2 button mice is:

- MouseSelect1
  Qt::LeftButton
- MouseSelect2
  Qt::RightButton
- MouseSelect3
  Qt::LeftButton + Qt::AltButton
- MouseSelect4
  Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
  Qt::RightButton + Qt::ShiftButton
- MouseSelect6
  Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

The default initialization for 1 button mice is:

- MouseSelect1
  Qt::LeftButton
- MouseSelect2
  Qt::LeftButton + Qt::ControlButton
- MouseSelect3
  Qt::LeftButton + Qt::AltButton
- MouseSelect4
  Qt::LeftButton + Qt::ShiftButton
- MouseSelect5
  Qt::LeftButton + Qt::ControlButton + Qt::ShiftButton
- MouseSelect6
  Qt::LeftButton + Qt::AltButton + Qt::ShiftButton

See also

initMousePattern()

12.27.3 Constructor & Destructor Documentation

12.27.3.1 QwtEventPattern::QwtEventPattern ()

Constructor

See also

MousePatternCode, KeyPatternCode

12.27.3.2 QwtEventPattern::~QwtEventPattern () [virtual]

Destructor.

12.27.4 Member Function Documentation

12.27.4.1 void QwtEventPattern::initKeyPattern ()

Set default mouse patterns.

See also

KeyPatternCode

12.27.4.2 void QwtEventPattern::initMousePattern (int *numButtons*)

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

Parameters

numButtons Number of mouse buttons (<= 3)

See also

MousePatternCode

12.27.4.3 bool QwtEventPattern::keyMatch (const KeyPattern & *pattern*, const QKeyEvent * *e*) const [protected, virtual]

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

pattern Key event pattern

e Key event

Returns

true if matches

See also

mouseMatch()

12.27.4.4 bool QwtEventPattern::keyMatch (uint *pattern*, const QKeyEvent * *e*) const

Compare a key event with an event pattern.

A key event matches the pattern when both have the same key value and in the state value the same key flags (Qt::KeyButtonMask) are set.

Parameters

pattern Index of the event pattern

e Key event

Returns

true if matches

See also

mouseMatch()

12.27.4.5 QwtArray< QwtEventPattern::KeyPattern > & QwtEventPattern::keyPattern ()

Return Key patterns.

12.27.4.6 const QwtArray< QwtEventPattern::KeyPattern > & QwtEventPattern::keyPattern () const

Return key patterns.

12.27.4.7 bool QwtEventPattern::mouseMatch (const MousePattern & *pattern*, const QMouseEvent * *e*) const [protected, virtual]

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(Qt::KeyButtonMask) are set.

Parameters

pattern Mouse event pattern

e Mouse event

Returns

true if matches

See also

keyMatch()

12.27.4.8 bool QwtEventPattern::mouseMatch (uint *pattern*, const QMouseEvent * *e*) const

Compare a mouse event with an event pattern.

A mouse event matches the pattern when both have the same button value and in the state value the same key flags(Qt::KeyButtonMask) are set.

Parameters

pattern Index of the event pattern

e Mouse event

Returns

true if matches

See also

keyMatch()

12.27.4.9 QwtArray< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern ()

Return ,ouse patterns.

12.27.4.10 const QwtArray< QwtEventPattern::MousePattern > & QwtEventPattern::mousePattern () const

Return mouse patterns.

12.27.4.11 void QwtEventPattern::setKeyPattern (const QwtArray< KeyPattern > & *pattern*)

Change the key event patterns.

12.27.4.12 void QwtEventPattern::setKeyPattern (uint *pattern*, int *key*, int *state* = `Qt::NoButton`)

Change one key pattern

Parameters

pattern Index of the pattern

key Key

state State

See also

QKeyEvent

12.27.4.13 void QwtEventPattern::setMousePattern (const QwtArray< MousePattern > & *pattern*)

Change the mouse event patterns.

12.27.4.14 void QwtEventPattern::setMousePattern (uint *pattern*, int *button*, int *state* = `Qt::NoButton`)

Change one mouse pattern

Parameters

pattern Index of the pattern

button Button

state State

See also

QMouseEvent

12.28 QwtIntervalData Class Reference

Series of samples of a value and an interval.

`#include <qwt_interval_data.h>`

Public Member Functions

- QwtIntervalData ()
- QwtIntervalData (const QwtArray< QwtDoubleInterval > &, const QwtArray< double > &)
- ~QwtIntervalData ()
- void setData (const QwtArray< QwtDoubleInterval > &, const QwtArray< double > &)
- size_t size () const
- const QwtDoubleInterval & interval (size_t i) const
- double value (size_t i) const
- QwtDoubleRect boundingRect () const

12.28.1 Detailed Description

Series of samples of a value and an interval. QwtIntervalData is a series of samples of a value and an interval. F.e. error bars are built from samples [x, y1-y2], while a histogram might consist of [x1-x2, y] samples.

12.28.2 Constructor & Destructor Documentation

12.28.2.1 QwtIntervalData::QwtIntervalData ()

Constructor.

12.28.2.2 QwtIntervalData::QwtIntervalData (const QwtArray< QwtDoubleInterval > & *intervals*, const QwtArray< double > & *values*)

Constructor.

12.28.2.3 QwtIntervalData::~QwtIntervalData ()

Destructor.

12.28.3 Member Function Documentation

12.28.3.1 QwtDoubleRect QwtIntervalData::boundingRect () const

Calculate the bounding rectangle of the samples

The x coordinates of the rectangle are built from the intervals, the y coordinates from the values.

Returns

Bounding rectangle

12.28.3.2 const QwtDoubleInterval & QwtIntervalData::interval (size_t *i*) const [inline]

Interval of a sample

Parameters

i Sample index

Returns

Interval

See also

value(), size()

12.28.3.3 void QwtIntervalData::setData (const QwtArray< QwtDoubleInterval > & *intervals*, const QwtArray< double > & *values*)

Assign samples.

12.28.3.4 size_t QwtIntervalData::size () const [inline]

Returns

Number of samples

12.28.3.5 double QwtIntervalData::value (size_t *i*) const [inline]

Value of a sample

Parameters

i Sample index

Returns

Value

See also

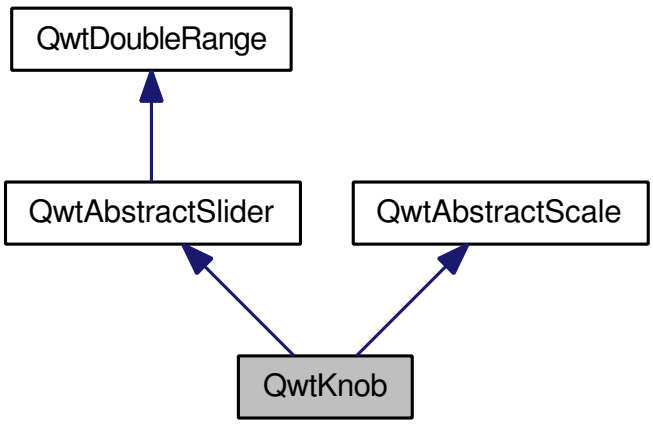
interval(), size()

12.29 QwtKnob Class Reference

The Knob Widget.

`#include <qwt_knob.h>`

Inheritance diagram for QwtKnob:



Public Types

- enum Symbol {
  Line,
  Dot }

Public Member Functions

- QwtKnob (QWidget *parent=NULL)
- virtual ~QwtKnob ()
- void setKnobWidth (int w)
- int knobWidth () const
- void setTotalAngle (double angle)

- double totalAngle () const
- void setBorderWidth (int bw)
- int borderWidth () const
- void setSymbol (Symbol)
- Symbol symbol () const
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- void setScaleDraw (QwtRoundScaleDraw ∗)
- const QwtRoundScaleDraw ∗ scaleDraw () const
- QwtRoundScaleDraw ∗ scaleDraw ()

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗e)
- virtual void resizeEvent (QResizeEvent ∗e)
- void draw (QPainter ∗p, const QRect &ur)
- void drawKnob (QPainter ∗p, const QRect &r)
- void drawMarker (QPainter ∗p, double arc, const QColor &c)

12.29.1 Detailed Description

The Knob Widget. The QwtKnob widget imitates look and behaviour of a volume knob on a radio. It contains a scale around the knob which is set up automatically or can be configured manually (see QwtAbstractScale). Automatic scrolling is enabled when the user presses a mouse button on the scale. For a description of signals, slots and other members, see QwtAbstractSlider.

See also

QwtAbstractSlider and QwtAbstractScale for the descriptions of the inherited members.

12.29.2 Member Enumeration Documentation

12.29.2.1 enum QwtKnob::Symbol

Symbol

See also

QwtKnob::QwtKnob()

12.29.3 Constructor & Destructor Documentation

12.29.3.1 QwtKnob::QwtKnob (QWidget ∗ *parent* = `NULL`) `[explicit]`

Constructor

Parameters

parent Parent widget

12.29.3.2 QwtKnob::~QwtKnob () [virtual]

Destructor.

12.29.4 Member Function Documentation

12.29.4.1 int QwtKnob::borderWidth () const

Return the border width.

12.29.4.2 void QwtKnob::draw (QPainter * *painter*, const QRect & *rect*) [protected]

Repaint the knob

Parameters

painter Painter

rect Update rectangle

12.29.4.3 void QwtKnob::drawKnob (QPainter * *painter*, const QRect & *r*) [protected]

Draw the knob.

Parameters

painter painter

r Bounding rectangle of the knob (without scale)

12.29.4.4 void QwtKnob::drawMarker (QPainter * *p*, double *arc*, const QColor & *c*) [protected]

Draw the marker at the knob's front.

Parameters

p Painter

arc Angle of the marker

c Marker color

12.29.4.5 int QwtKnob::knobWidth () const

Return the width of the knob.

12.29.4.6 QSize QwtKnob::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

The return value of QwtKnob::minimumSizeHint() depends on the font and the scale.

12.29.4.7 void QwtKnob::paintEvent (QPaintEvent * *e*) [protected, virtual]

Repaint the knob

Parameters

e Paint event

12.29.4.8 void QwtKnob::resizeEvent (QResizeEvent * *e*) [protected, virtual]

Qt Resize Event

12.29.4.9 QwtRoundScaleDraw * QwtKnob::scaleDraw ()

Returns

the scale draw of the knob

See also

setScaleDraw()

12.29.4.10 const QwtRoundScaleDraw * QwtKnob::scaleDraw () const

Returns

the scale draw of the knob

See also

setScaleDraw()

12.29.4.11 void QwtKnob::setBorderWidth (int *bw*)

Set the knob's border width.

Parameters

bw new border width

12.29.4.12 void QwtKnob::setKnobWidth (int *w*)

Change the knob's width.

The specified width must be >= 5, or it will be clipped.

Parameters

w New width

12.29.4.13 void QwtKnob::setScaleDraw (QwtRoundScaleDraw * *scaleDraw*)

Change the scale draw of the knob

For changing the labels of the scales, it is necessary to derive from QwtRoundScaleDraw and overload QwtRoundScaleDraw::label().

See also

scaleDraw()

12.29.4.14 void QwtKnob::setSymbol (QwtKnob::Symbol *s*)

Set the symbol of the knob.

See also

symbol()

12.29.4.15 void QwtKnob::setTotalAngle (double *angle*)

Set the total angle by which the knob can be turned.

Parameters

angle Angle in degrees.

The default angle is 270 degrees. It is possible to specify an angle of more than 360 degrees so that the knob can be turned several times around its axis.

12.29.4.16 QSize QwtKnob::sizeHint () const [virtual]

Returns

minimumSizeHint()

12.29.4.17 QwtKnob::Symbol QwtKnob::symbol () const

Returns

symbol of the knob

See also

setSymbol()

12.29.4.18 double QwtKnob::totalAngle () const

Return the total angle.

12.30 QwtLegend Class Reference

The legend widget.

`#include <qwt_legend.h>`

Public Types

- enum LegendDisplayPolicy {
  NoIdentifier = 0,
  FixedIdentifier = 1,
  AutoIdentifier = 2 }
- enum LegendItemMode {
  ReadOnlyItem,
  ClickableItem,
  CheckableItem }

Public Member Functions

- QwtLegend (QWidget *parent=NULL)
- virtual ~QwtLegend ()
- void setDisplayPolicy (LegendDisplayPolicy policy, int mode)
- LegendDisplayPolicy displayPolicy () const
- void setItemMode (LegendItemMode)
- LegendItemMode itemMode () const
- int identifierMode () const
- QWidget * contentsWidget ()
- const QWidget * contentsWidget () const
- void insert (const QwtLegendItemManager *, QWidget *)
- void remove (const QwtLegendItemManager *)
- QWidget * find (const QwtLegendItemManager *) const

- QwtLegendItemManager * find (const QWidget *) const
- virtual QList< QWidget * > legendItems () const
- void clear ()
- bool isEmpty () const
- uint itemCount () const
- virtual bool eventFilter (QObject *, QEvent *)
- virtual QSize sizeHint () const
- virtual int heightForWidth (int w) const
- QScrollBar * horizontalScrollBar () const
- QScrollBar * verticalScrollBar () const

Protected Member Functions

- virtual void resizeEvent (QResizeEvent *)
- virtual void layoutContents ()

12.30.1 Detailed Description

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

See also

QwtLegendItem, QwtLegendItemManager QwtPlot

12.30.2 Member Enumeration Documentation

12.30.2.1 enum QwtLegend::LegendDisplayPolicy

Display policy.

- NoIdentifier
  The client code is responsible how to display of each legend item. The Qwt library will not interfere.
- FixedIdentifier
  All legend items are displayed with the QwtLegendItem::IdentifierMode to be passed in 'mode'.
- AutoIdentifier
  Each legend item is displayed with a mode that is a bitwise or of
  - QwtLegendItem::ShowLine (if its curve is drawn with a line) and
  - QwtLegendItem::ShowSymbol (if its curve is drawn with symbols) and
  - QwtLegendItem::ShowText (if the has a title).

Default is AutoIdentifier.

See also

setDisplayPolicy(), displayPolicy(), QwtLegendItem::IdentifierMode

12.30.2.2 enum QwtLegend::LegendItemMode

Interaction mode for the legend items.

- ReadOnlyItem
  The legend item is not interactive, like a label
- ClickableItem
  The legend item is clickable, like a push button
- CheckableItem
  The legend item is checkable, like a checkable button

Default is ReadOnlyItem.

See also

setItemMode(), itemMode(), QwtLegendItem::IdentifierMode QwtLegendItem::clicked(), QwtLegendItem::checked(), QwtPlot::legendClicked(), QwtPlot::legendChecked()

12.30.3 Constructor & Destructor Documentation

12.30.3.1 QwtLegend::QwtLegend (QWidget * *parent* = NULL) [explicit]

Constructor

Parameters

parent Parent widget

12.30.3.2 QwtLegend::~QwtLegend () [virtual]

Destructor.

12.30.4 Member Function Documentation

12.30.4.1 void QwtLegend::clear ()

Remove all items.

12.30.4.2 const QWidget * QwtLegend::contentsWidget () const

The contents widget is the only child of the viewport() and the parent widget of all legend items.

12.30.4.3 QWidget * QwtLegend::contentsWidget ()

The contents widget is the only child of the viewport() and the parent widget of all legend items.

12.30.4.4 QwtLegend::LegendDisplayPolicy QwtLegend::displayPolicy () const

Returns

the legend display policy. Default is LegendDisplayPolicy::Auto.

See also

setDisplayPolicy(), LegendDisplayPolicy

12.30.4.5 bool QwtLegend::eventFilter (QObject * *o*, QEvent * *e*) [virtual]

Filter layout related events of QwtLegend::contentsWidget().

Parameters

o Object to be filtered

e Event

12.30.4.6 QwtLegendItemManager * QwtLegend::find (const QWidget * *legendItem*) const

Find the widget that represents a plot item

Parameters

legendItem Legend item

Returns

Widget on the legend, or NULL

12.30.4.7 QWidget * QwtLegend::find (const QwtLegendItemManager * *plotItem*) const

Find the widget that represents a plot item

Parameters

plotItem Plot item

Returns

Widget on the legend, or NULL

12.30.4.8 int QwtLegend::heightForWidth (int *width*) const [virtual]

Returns

The preferred height, for the width w.

Parameters

width Width

12.30.4.9 QScrollBar * QwtLegend::horizontalScrollBar () const

Returns

Horizontal scrollbar

See also

verticalScrollBar()

12.30.4.10 int QwtLegend::identifierMode () const

Returns

the IdentifierMode to be used in combination with LegendDisplayPolicy::Fixed.

Default is ShowLine | ShowSymbol | ShowText.

12.30.4.11 void QwtLegend::insert (const QwtLegendItemManager * *plotItem*, QWidget * *legendItem*)

Insert a new item for a plot item

Parameters

plotItem Plot item

legendItem New legend item

Note

The parent of item will be changed to QwtLegend::contentsWidget()

12.30.4.12 bool QwtLegend::isEmpty () const

Return true, if there are no legend items.

12.30.4.13 uint QwtLegend::itemCount () const

Return the number of legend items.

12.30.4.14 QwtLegend::LegendItemMode QwtLegend::itemMode () const

See also

LegendItemMode

12.30.4.15 void QwtLegend::layoutContents () [protected, virtual]

Adjust contents widget and item layout to the size of the viewport().

12.30.4.16 QList< QWidget * > QwtLegend::legendItems () const [virtual]

Return a list of all legend items.

12.30.4.17 void QwtLegend::remove (const QwtLegendItemManager * *plotItem*)

Find the corresponding item for a plotItem and remove it from the item list.

Parameters

plotItem Plot item

12.30.4.18 void QwtLegend::resizeEvent (QResizeEvent * *e*) [protected, virtual]

Resize event

Parameters

e Resize event

12.30.4.19 void QwtLegend::setDisplayPolicy (LegendDisplayPolicy *policy*, int *mode*)

Set the legend display policy to:

Parameters

policy Legend display policy

mode Identifier mode (or'd ShowLine, ShowSymbol, ShowText)

See also

displayPolicy(), LegendDisplayPolicy

12.30.4.20 void QwtLegend::setItemMode (LegendItemMode *mode*)

See also

LegendItemMode

12.30.4.21 QSize QwtLegend::sizeHint () const [virtual]

Return a size hint.

12.30.4.22 QScrollBar * QwtLegend::verticalScrollBar () const

Returns

Vertical scrollbar

See also

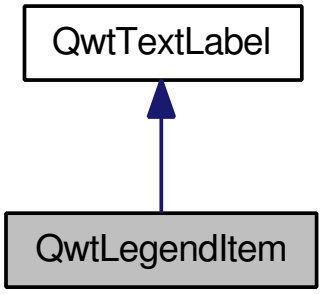
horizontalScrollBar()

12.31 QwtLegendItem Class Reference

A legend label.

`#include <qwt_legend_item.h>`

Inheritance diagram for QwtLegendItem:



Public Types

- enum IdentifierMode {
  NoIdentifier = 0,
  ShowLine = 1,
  ShowSymbol = 2,
  ShowText = 4 }

Public Slots

- void setChecked (bool on)

Signals

- void clicked ()
- void pressed ()
- void released ()
- void checked (bool)

Public Member Functions

- QwtLegendItem (QWidget *parent=0)
- QwtLegendItem (const QwtSymbol &, const QPen &, const QwtText &, QWidget *parent=0)
- virtual ~QwtLegendItem ()
- virtual void setText (const QwtText &)
- void setItemMode (QwtLegend::LegendItemMode)
- QwtLegend::LegendItemMode itemMode () const
- void setIdentifierMode (int)
- int identifierMode () const
- void setIdentifierWidth (int width)
- int identifierWidth () const
- void setSpacing (int spacing)
- int spacing () const
- void setSymbol (const QwtSymbol &)
- const QwtSymbol & symbol () const
- void setCurvePen (const QPen &)
- const QPen & curvePen () const
- virtual void drawIdentifier (QPainter *, const QRect &) const
- virtual void drawItem (QPainter *p, const QRect &) const
- virtual QSize sizeHint () const
- bool isChecked () const

Protected Member Functions

- void setDown (bool)
- bool isDown () const
- virtual void paintEvent (QPaintEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void drawText (QPainter *, const QRect &)

12.31.1 Detailed Description

A legend label. QwtLegendItem represents a curve on a legend. It displays an curve identifier with an explaining text. The identifier might be a combination of curve symbol and line. In readonly mode it behaves like a label, otherwise like an unstylish push button.

See also

QwtLegend, QwtPlotCurve

12.31.2 Member Enumeration Documentation

12.31.2.1 enum QwtLegendItem::IdentifierMode

Identifier mode.

Default is ShowLine | ShowText

See also

identifierMode(), setIdentifierMode()

12.31.3 Constructor & Destructor Documentation

12.31.3.1 QwtLegendItem::QwtLegendItem (QWidget * *parent* = 0) [explicit]

Parameters

parent Parent widget

12.31.3.2 QwtLegendItem::QwtLegendItem (const QwtSymbol & *symbol*, const QPen & *curvePen*, const QwtText & *text*, QWidget * *parent* = 0) [explicit]

Parameters

symbol Curve symbol

curvePen Curve pen

text Label text

parent Parent widget

12.31.3.3 QwtLegendItem::~QwtLegendItem () [virtual]

Destructor.

12.31.4 Member Function Documentation

12.31.4.1 void QwtLegendItem::checked (bool) [signal]

Signal, when the legend item has been toggled.

12.31.4.2 void QwtLegendItem::clicked () [signal]

Signal, when the legend item has been clicked.

12.31.4.3 const QPen & QwtLegendItem::curvePen () const

Returns

The curve pen.

See also

setCurvePen()

12.31.4.4 void QwtLegendItem::drawIdentifier (QPainter * *painter*, const QRect & *rect*) const [virtual]

Paint the identifier to a given rect.

Parameters

painter Painter

rect Rect where to paint

12.31.4.5 void QwtLegendItem::drawItem (QPainter * *painter*, const QRect & *rect*) const [virtual]

Draw the legend item to a given rect.

Parameters

painter Painter

rect Rect where to paint the button

12.31.4.6 void QwtLegendItem::drawText (QPainter * *painter*, const QRect & *textRect*) [protected, virtual]

Redraw the text.

Reimplemented from QwtTextLabel.

12.31.4.7 int QwtLegendItem::identifierMode () const

Or'd values of IdentifierMode.

See also

setIdentifierMode(), IdentifierMode

12.31.4.8 int QwtLegendItem::identifierWidth () const

Return the width of the identifier

See also

setIdentifierWidth()

12.31.4.9 bool QwtLegendItem::isChecked () const

Return true, if the item is checked.

12.31.4.10 bool QwtLegendItem::isDown () const [protected]

Return true, if the item is down.

12.31.4.11 QwtLegend::LegendItemMode QwtLegendItem::itemMode () const

Return the item mode

See also

setItemMode()

12.31.4.12 void QwtLegendItem::keyPressEvent (QKeyEvent * *e*) [protected, virtual]

Handle key press events.

12.31.4.13 void QwtLegendItem::keyReleaseEvent (QKeyEvent * *e*) [protected, virtual]

Handle key release events.

12.31.4.14 void QwtLegendItem::mousePressEvent (QMouseEvent * *e*) [protected, virtual]

Handle mouse press events.

12.31.4.15 void QwtLegendItem::mouseReleaseEvent (QMouseEvent * *e*) [protected, virtual]

Handle mouse release events.

12.31.4.16 void QwtLegendItem::paintEvent (QPaintEvent * *e*) [protected, virtual]

Paint event.

Reimplemented from QwtTextLabel.

12.31.4.17 void QwtLegendItem::pressed () [signal]

Signal, when the legend item has been pressed.

12.31.4.18 void QwtLegendItem::released () [signal]

Signal, when the legend item has been relased.

12.31.4.19 void QwtLegendItem::setChecked (bool *on*) [slot]

Check/Uncheck a the item

Parameters

on check/uncheck

See also

setItemMode()

12.31.4.20 void QwtLegendItem::setCurvePen (const QPen & *pen*)

Set curve pen.

Parameters

pen Curve pen

See also

curvePen()

12.31.4.21 void QwtLegendItem::setDown (bool *down*) [protected]

Set the item being down.

12.31.4.22 void QwtLegendItem::setIdentifierMode (int *mode*)

Set identifier mode. Default is ShowLine | ShowText.

Parameters

mode Or'd values of IdentifierMode

See also

identifierMode()

12.31.4.23 void QwtLegendItem::setIdentifierWidth (int *width*)

Set the width for the identifier Default is 8 pixels

Parameters

width New width

See also

identifierMode(), identifierWidth()

12.31.4.24 void QwtLegendItem::setItemMode (QwtLegend::LegendItemMode *mode*)

Set the item mode The default is QwtLegend::ReadOnlyItem

Parameters

mode Item mode

See also

itemMode()

12.31.4.25 void QwtLegendItem::setSpacing (int *spacing*)

Change the spacing

Parameters

spacing Spacing

See also

spacing(), identifierWidth(), QwtTextLabel::margin()

12.31.4.26 void QwtLegendItem::setSymbol (const QwtSymbol & *symbol*)

Set curve symbol.

Parameters

symbol Symbol

See also

symbol()

12.31.4.27 void QwtLegendItem::setText (const QwtText & *text*) [virtual]

Set the text to the legend item

Parameters

text Text label

See also

QwtTextLabel::text()

Reimplemented from QwtTextLabel.

12.31.4.28 QSize QwtLegendItem::sizeHint () const [virtual]

Return a size hint.

Reimplemented from QwtTextLabel.

12.31.4.29 int QwtLegendItem::spacing () const

Return the spacing

See also

setSpacing(), identifierWidth(), QwtTextLabel::margin()

12.31.4.30 const QwtSymbol & QwtLegendItem::symbol () const

Returns

The curve symbol.

See also

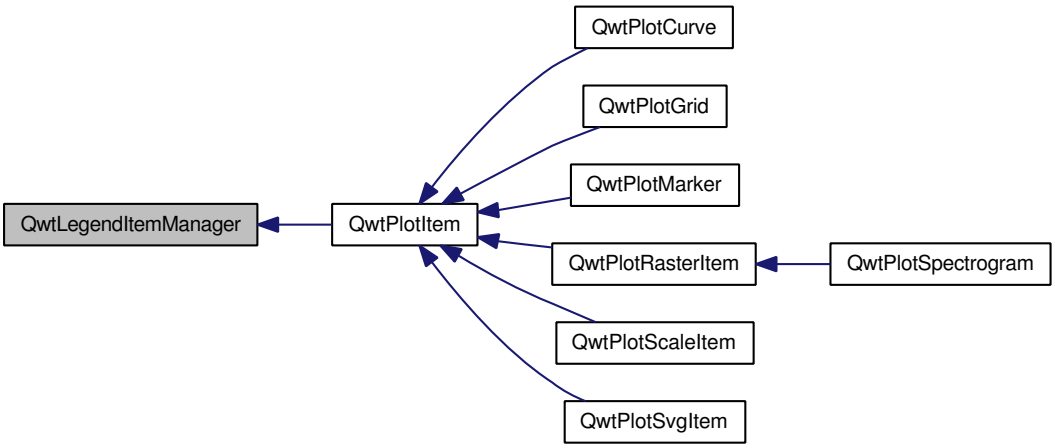
setSymbol()

12.32 QwtLegendItemManager Class Reference

Abstract API to bind plot items to the legend.

`#include <qwt_legend_itemmanager.h>`

Inheritance diagram for QwtLegendItemManager:



Public Member Functions

- QwtLegendItemManager ()
- virtual ~QwtLegendItemManager ()
- virtual void updateLegend (QwtLegend *legend) const =0
- virtual QWidget * legendItem () const =0

12.32.1 Detailed Description

Abstract API to bind plot items to the legend.

12.32.2 Constructor & Destructor Documentation

12.32.2.1 QwtLegendItemManager::QwtLegendItemManager () [inline]

Constructor.

12.32.2.2 virtual QwtLegendItemManager::~QwtLegendItemManager () [inline, virtual]

Destructor.

12.32.3 Member Function Documentation

12.32.3.1 virtual QWidget* QwtLegendItemManager::legendItem () const [pure virtual]

Allocate the widget that represents the item on the legend

Returns

Allocated widget

See also

updateLegend() QwtLegend()

Implemented in QwtPlotItem.

12.32.3.2 virtual void QwtLegendItemManager::updateLegend (QwtLegend * *legend*) const [pure virtual]

Update the widget that represents the item on the legend

Parameters

legend Legend

See also

legendItem()

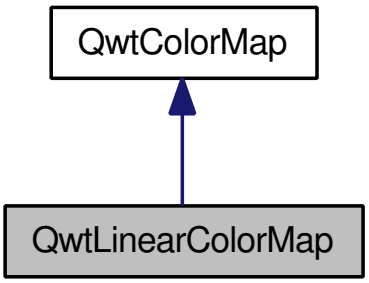
Implemented in QwtPlotCurve, and QwtPlotItem.

12.33 QwtLinearColorMap Class Reference

QwtLinearColorMap builds a color map from color stops.

`#include <qwt_color_map.h>`

Inheritance diagram for QwtLinearColorMap:



Public Types

- enum Mode {
  FixedColors,
  ScaledColors }

Public Member Functions

- QwtLinearColorMap (QwtColorMap::Format=QwtColorMap::RGB)
- QwtLinearColorMap (const QColor &from, const QColor &to, QwtColorMap::Format=QwtColorMap::RGB)
- QwtLinearColorMap (const QwtLinearColorMap &)
- virtual ~QwtLinearColorMap ()
- QwtLinearColorMap & operator= (const QwtLinearColorMap &)
- virtual QwtColorMap * copy () const
- void setMode (Mode)
- Mode mode () const
- void setColorInterval (const QColor &color1, const QColor &color2)
- void addColorStop (double value, const QColor &)
- QwtArray< double > colorStops () const
- QColor color1 () const
- QColor color2 () const
- virtual QRgb rgb (const QwtDoubleInterval &, double value) const
- virtual unsigned char colorIndex (const QwtDoubleInterval &, double value) const

12.33.1 Detailed Description

QwtLinearColorMap builds a color map from color stops. A color stop is a color at a specific position. The valid range for the positions is [0.0, 1.0]. When mapping a value into a color it is translated into this interval. If mode() == FixedColors the color is calculated from the next lower color stop. If mode() == ScaledColors the color is calculated by interpolating the colors of the adjacent stops.

12.33.2 Member Enumeration Documentation

12.33.2.1 enum QwtLinearColorMap::Mode

Mode of color map

See also

setMode(), mode()

12.33.3 Constructor & Destructor Documentation

12.33.3.1 QwtLinearColorMap::QwtLinearColorMap (QwtColorMap::Format *format* = `QwtColorMap::RGB`)

Build a color map with two stops at 0.0 and 1.0. The color at 0.0 is Qt::blue, at 1.0 it is Qt::yellow.

Parameters

format Preferred format of the color map

12.33.3.2 QwtLinearColorMap::QwtLinearColorMap (const QColor & *color1*, const QColor & *color2*, QwtColorMap::Format *format* = `QwtColorMap::RGB`)

Build a color map with two stops at 0.0 and 1.0.

Parameters

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 of the coor map

12.33.3.3 QwtLinearColorMap::QwtLinearColorMap (const QwtLinearColorMap & *other*)

Copy constructor.

12.33.3.4 QwtLinearColorMap::~QwtLinearColorMap () `[virtual]`

Destructor.

12.33.4 Member Function Documentation

12.33.4.1 void QwtLinearColorMap::addColorStop (double *value*, const QColor & *color*)

Add a color stop

The value has to be in the range [0.0, 1.0]. F.e. a stop at position 17.0 for a range [10.0,20.0] must be passed as: (17.0 - 10.0) / (20.0 - 10.0)

Parameters

value Value between [0.0, 1.0]

color Color stop

12.33.4.2 QColor QwtLinearColorMap::color1 () const

Returns

the first color of the color range

See also

setColorInterval()

12.33.4.3 QColor QwtLinearColorMap::color2 () const

Returns

the second color of the color range

See also

setColorInterval()

12.33.4.4 unsigned char QwtLinearColorMap::colorIndex (const QwtDoubleInterval & *interval*, double *value*) const [virtual]

Map a value of a given interval into a color index, between 0 and 255

Parameters

interval Range for all values

value Value to map into a color index

Implements QwtColorMap.

12.33.4.5 QwtArray< double > QwtLinearColorMap::colorStops () const

Return all positions of color stops in increasing order

12.33.4.6 QwtColorMap * QwtLinearColorMap::copy () const [virtual]

Clone the color map.

Implements QwtColorMap.

12.33.4.7 QwtLinearColorMap::Mode QwtLinearColorMap::mode () const

Returns

Mode of the color map

See also

setMode()

12.33.4.8 QwtLinearColorMap & QwtLinearColorMap::operator= (const QwtLinearColorMap & *other*)

Assignment operator.

12.33.4.9 QRgb QwtLinearColorMap::rgb (const QwtDoubleInterval & *interval*, double *value*) const [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

Implements QwtColorMap.

12.33.4.10 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()

12.33.4.11 void QwtLinearColorMap::setMode (Mode *mode*)

Set the mode of the color map.

FixedColors means the color is calculated from the next lower color stop. ScaledColors means the color is calculated by interpolating the colors of the adjacent stops.

See also

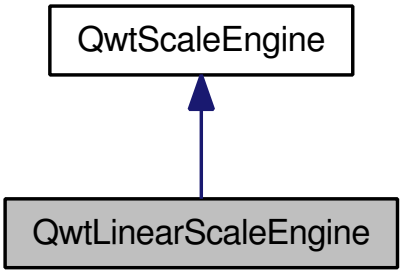
mode()

12.34 QwtLinearScaleEngine Class Reference

A scale engine for linear scales.

```
#include <qwt_scale_engine.h>
```

Inheritance diagram for QwtLinearScaleEngine:



Public Member Functions

- virtual void autoScale (int maxSteps, double &x1, double &x2, double &stepSize) const
- virtual QwtScaleDiv divideScale (double x1, double x2, int numMajorSteps, int numMinorSteps, double stepSize=0.0) const
- virtual QwtScaleTransformation * transformation () const

Protected Member Functions

- QwtDoubleInterval align (const QwtDoubleInterval &, double stepSize) const

12.34.1 Detailed Description

A scale engine for linear scales. The step size will fit into the pattern $\{1, 2, 5\} \cdot 10^n$, where n is an integer.

12.34.2 Member Function Documentation

12.34.2.1 QwtDoubleInterval QwtLinearScaleEngine::align (const QwtDoubleInterval & *interval*, double *stepSize*) const [protected]

Align an interval to a step size.

The limits of an interval are aligned that both are integer multiples of the step size.

Parameters

interval Interval

stepSize Step size

Returns

Aligned interval

12.34.2.2 void QwtLinearScaleEngine::autoScale (int *maxNumSteps*, double & *x1*, double & *x2*, double & *stepSize*) const [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.

12.34.2.3 QwtScaleDiv QwtLinearScaleEngine::divideScale (double *x1*, double *x2*, int *maxMajSteps*, int *maxMinSteps*, double *stepSize* = 0.0) const [virtual]

Calculate a scale division.

Parameters

x1 First interval limit

x2 Second interval limit

maxMajSteps Maximum for the number of major steps

maxMinSteps Maximum number of minor steps

stepSize Step size. If stepSize == 0, the scaleEngine calculates one.

See also

QwtScaleEngine::stepSize(), QwtScaleEngine::subDivide()

Implements QwtScaleEngine.

12.34.2.4 QwtScaleTransformation * QwtLinearScaleEngine::transformation () const [virtual]

Return a transformation, for linear scales

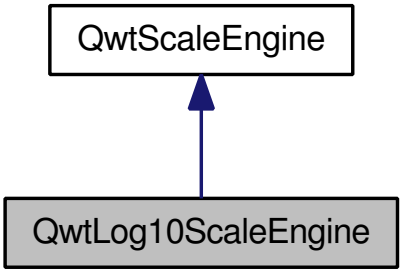
Implements QwtScaleEngine.

12.35 QwtLog10ScaleEngine Class Reference

A scale engine for logarithmic (base 10) scales.

`#include <qwt_scale_engine.h>`

Inheritance diagram for QwtLog10ScaleEngine:



Public Member Functions

- virtual void autoScale (int maxSteps, double &x1, double &x2, double &stepSize) const
- virtual QwtScaleDiv divideScale (double x1, double x2, int numMajorSteps, int numMinorSteps, double stepSize=0.0) const
- virtual QwtScaleTransformation * transformation () const

Protected Member Functions

- QwtDoubleInterval log10 (const QwtDoubleInterval &) const
- QwtDoubleInterval pow10 (const QwtDoubleInterval &) const

12.35.1 Detailed Description

A scale engine for logarithmic (base 10) scales. The step size is measured in *decades* and the major step size will be adjusted to fit the pattern $\{1, 2, 3, 5\} \cdot 10^n$, where n is a natural number including zero.

Warning

the step size as well as the margins are measured in *decades*.

12.35.2 Member Function Documentation

12.35.2.1 void QwtLog10ScaleEngine::autoScale (int *maxNumSteps*, double & *x1*, double & *x2*, double & *stepSize*) const [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.

12.35.2.2 QwtScaleDiv QwtLog10ScaleEngine::divideScale (double *x1*, double *x2*, int *maxMajSteps*, int *maxMinSteps*, double *stepSize* = 0.0) const [virtual]

Calculate a scale division.

Parameters

x1 First interval limit

x2 Second interval limit

maxMajSteps Maximum for the number of major steps

maxMinSteps Maximum number of minor steps

stepSize Step size. If stepSize == 0, the scaleEngine calculates one.

See also

QwtScaleEngine::stepSize(), QwtLog10ScaleEngine::subDivide()

Implements QwtScaleEngine.

12.35.2.3 QwtDoubleInterval QwtLog10ScaleEngine::log10 (const QwtDoubleInterval & *interval*) const [protected]

Return the interval [log10(interval.minValue(), log10(interval.maxValue]

12.35.2.4 QwtDoubleInterval QwtLog10ScaleEngine::pow10 (const QwtDoubleInterval & *interval*) const [protected]

Return the interval [pow10(interval.minValue(), pow10(interval.maxValue]

12.35.2.5 QwtScaleTransformation * QwtLog10ScaleEngine::transformation () const [virtual]

Return a transformation, for logarithmic (base 10) scales

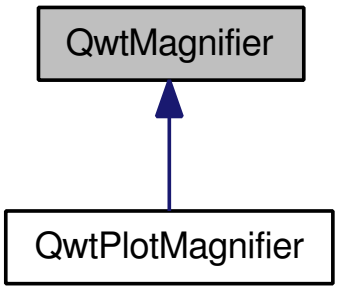
Implements QwtScaleEngine.

12.36 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 ()
- QWidget ∗ parentWidget ()
- const QWidget ∗ parentWidget () const
- void setEnabled (bool)
- bool isEnabled () const
- void setMouseFactor (double)
- double mouseFactor () const
- void setMouseButton (int button, int buttonState=Qt::NoButton)
- void getMouseButton (int &button, int &buttonState) const
- void setWheelFactor (double)
- double wheelFactor () const
- void setWheelButtonState (int buttonState)
- int wheelButtonState () const
- void setKeyFactor (double)
- double keyFactor () const
- void setZoomInKey (int key, int modifiers)
- void getZoomInKey (int &key, int &modifiers) const
- void setZoomOutKey (int key, int modifiers)
- void getZoomOutKey (int &key, int &modifiers) const
- virtual bool eventFilter (QObject ∗, QEvent ∗)

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 ∗)

12.36.1 Detailed Description

QwtMagnifier provides zooming, by magnifying in steps. Using QwtMagnifier a plot can be zoomed in/out in steps using keys, the mouse wheel or moving a mouse button in vertical direction.

12.36.2 Constructor & Destructor Documentation

12.36.2.1 QwtMagnifier::QwtMagnifier (QWidget * *parent*) [explicit]

Constructor

Parameters

parent Widget to be magnified

12.36.2.2 QwtMagnifier::~QwtMagnifier () [virtual]

Destructor.

12.36.3 Member Function Documentation

12.36.3.1 bool QwtMagnifier::eventFilter (QObject * *o*, QEvent * *e*) [virtual]

Event filter.

When isEnabled() the mouse events of the observed widget are filtered.

See also

widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent() widgetKeyReleaseEvent()

12.36.3.2 void QwtMagnifier::getMouseButton (int & *button*, int & *buttonState*) const

See also

setMouseButton()

12.36.3.3 void QwtMagnifier::getZoomInKey (int & *key*, int & *modifiers*) const

See also

setZoomInKey()

12.36.3.4 void QwtMagnifier::getZoomOutKey (int & *key*, int & *modifiers*) const

See also

setZoomOutKey()

12.36.3.5 bool QwtMagnifier::isEnabled () const

Returns

true when enabled, false otherwise

See also

setEnabled(), eventFilter()

12.36.3.6 double QwtMagnifier::keyFactor () const

Returns

Key factor

See also

setKeyFactor()

12.36.3.7 double QwtMagnifier::mouseFactor () const

Returns

Mouse factor

See also

setMouseFactor()

12.36.3.8 const QWidget * QwtMagnifier::parentWidget () const

Returns

Parent widget, where the rescaling happens

12.36.3.9 QWidget * QwtMagnifier::parentWidget ()

Returns

Parent widget, where the rescaling happens

12.36.3.10 virtual void QwtMagnifier::rescale (double *factor*) [protected, pure virtual]

Rescale the parent widget

Parameters

factor Scale factor

Implemented in QwtPlotMagnifier.

12.36.3.11 void QwtMagnifier::setEnabled (bool *on*)

En/disable the magnifier.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

on true or false

See also

isEnabled(), eventFilter()

12.36.3.12 void QwtMagnifier::setKeyFactor (double *factor*)

Change the key factor.

The key factor defines the ratio between the current range on the parent widget and the zoomed range for each key press of the zoom in/out keys. The default value is 0.9.

Parameters

factor Key factor

See also

keyFactor(), setZoomInKey(), setZoomOutKey(), setWheelFactor, setMouseFactor()

12.36.3.13 void QwtMagnifier::setMouseButton (int *button*, int *buttonState* = Qt::NoButton)

Assign the mouse button, that is used for zooming in/out. The default value is Qt::RightButton.

Parameters

button Button

buttonState Button state

See also

getMouseButton()

12.36.3.14 void QwtMagnifier::setMouseFactor (double *factor*)

Change the mouse factor.

The mouse factor defines the ratio between the current range on the parent widget and the zoomed range for each vertical mouse movement. The default value is 0.95.

Parameters

factor Wheel factor

See also

mouseFactor(), setMouseButton(), setWheelFactor(), setKeyFactor()

12.36.3.15 void QwtMagnifier::setWheelButtonState (int *buttonState*)

Assign a mandatory button state for zooming in/out using the wheel. The default button state is Qt::NoButton.

Parameters

buttonState Button state

See also

wheelButtonState()

12.36.3.16 void QwtMagnifier::setWheelFactor (double *factor*)

Change the wheel factor.

The wheel factor defines the ratio between the current range on the parent widget and the zoomed range for each step of the wheel. The default value is 0.9.

Parameters

factor Wheel factor

See also

wheelFactor(), setWheelButtonState(), setMouseFactor(), setKeyFactor()

12.36.3.17 void QwtMagnifier::setZoomInKey (int *key*, int *modifiers*)

Assign the key, that is used for zooming in. The default combination is Qt::Key_Plus + Qt::NoModifier.

Parameters

key

modifiers

See also

getZoomInKey(), setZoomOutKey()

12.36.3.18 void QwtMagnifier::setZoomOutKey (int *key*, int *modifiers*)

Assign the key, that is used for zooming out. The default combination is Qt::Key_Minus + Qt::NoModifier.

Parameters

key
modifiers

See also

getZoomOutKey(), setZoomOutKey()

12.36.3.19 int QwtMagnifier::wheelButtonState () const

Returns

Wheel button state

See also

setWheelButtonState()

12.36.3.20 double QwtMagnifier::wheelFactor () const

Returns

Wheel factor

See also

setWheelFactor()

12.36.3.21 void QwtMagnifier::widgetKeyPressEvent (QKeyEvent * *ke*) [protected, virtual]

Handle a key press event for the observed widget.

Parameters

ke Key event

See also

eventFilter(), widgetKeyReleaseEvent()

12.36.3.22 void QwtMagnifier::widgetKeyReleaseEvent (QKeyEvent *) [protected, virtual]

Handle a key release event for the observed widget.

Parameters

ke Key event

See also

eventFilter(), widgetKeyReleaseEvent()

12.36.3.23 void QwtMagnifier::widgetMouseMoveEvent (QMouseEvent * *me*) [protected, virtual]

Handle a mouse move event for the observed widget.

Parameters

me Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(),

12.36.3.24 void QwtMagnifier::widgetMousePressEvent (QMouseEvent * *me*) [protected, virtual]

Handle a mouse press event for the observed widget.

Parameters

me Mouse event

See also

eventFilter(), widgetMouseReleaseEvent(), widgetMouseMoveEvent()

12.36.3.25 void QwtMagnifier::widgetMouseReleaseEvent (QMouseEvent *) [protected, virtual]

Handle a mouse release event for the observed widget.

See also

eventFilter(), widgetMousePressEvent(), widgetMouseMoveEvent(),

12.36.3.26 void QwtMagnifier::widgetWheelEvent (QWheelEvent * *we*) [protected, virtual]

Handle a wheel event for the observed widget.

Parameters

we Wheel event

See also

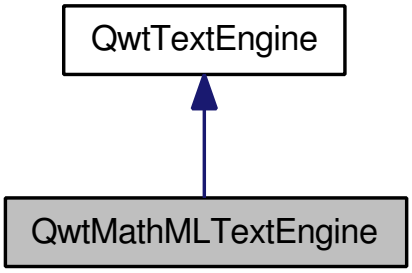
eventFilter()

12.37 QwtMathMLTextEngine Class Reference

Text Engine for the MathML renderer of the Qt solutions package.

`#include <qwt_mathml_text_engine.h>`

Inheritance diagram for QwtMathMLTextEngine:



Public Member Functions

- QwtMathMLTextEngine ()
- virtual ~QwtMathMLTextEngine ()
- virtual int heightForWidth (const QFont &font, int flags, const QString &text, int width) const
- virtual QSize textSize (const QFont &font, int flags, const QString &text) const
- virtual void draw (QPainter *painter, const QRect &rect, int flags, const QString &text) const
- virtual bool mightRender (const QString &) const
- virtual void textMargins (const QFont &, const QString &, int &left, int &right, int &top, int &bottom) const

12.37.1 Detailed Description

Text Engine for the MathML renderer of the Qt solutions package. The Qt Solution package includes a renderer for MathML http://www.trolltech.com/products/qt/addon/solutions/catalog/4/Widgets/qtmmlwidget that is available for owners of a commercial Qt license. You need a version >= 2.1, that is only available for Qt4.

To enable MathML support the following code needs to be added to the application:

```
#include <qwt_mathml_text_engine.h>

QwtText::setTextEngine(QwtText::MathMLText, new QwtMathMLTextEngine());
```

See also

QwtTextEngine, QwtText::setTextEngine

Warning

Unfortunately the MathML renderer doesn't support rotating of texts.

12.37.2 Constructor & Destructor Documentation

12.37.2.1 QwtMathMLTextEngine::QwtMathMLTextEngine ()

Constructor.

12.37.2.2 QwtMathMLTextEngine::~QwtMathMLTextEngine () [virtual]

Destructor.

12.37.3 Member Function Documentation

12.37.3.1 void QwtMathMLTextEngine::draw (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) const [virtual]

Draw the text in a clipping rectangle

Parameters

painter Painter

rect Clipping rectangle

flags Bitwise OR of the flags like in for QPainter::drawText

text Text to be rendered

Implements QwtTextEngine.

12.37.3.2 int QwtMathMLTextEngine::heightForWidth (const QFont & *font*, int *flags*, const QString & *text*, int *width*) const [virtual]

Find the height for a given width

Parameters

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.

12.37.3.3 bool QwtMathMLTextEngine::mightRender (const QString & *text*) const [virtual]

Test if a string can be rendered by QwtMathMLTextEngine

Parameters

text Text to be tested

Returns

true, if text begins with "<math>".

Implements QwtTextEngine.

12.37.3.4 void QwtMathMLTextEngine::textMargins (const QFont &, const QString &, int & *left*, int & *right*, int & *top*, int & *bottom*) const [virtual]

Return margins around the texts

Parameters

left Return 0

right Return 0

top Return 0

bottom Return 0

Implements QwtTextEngine.

12.37.3.5 QSize QwtMathMLTextEngine::textSize (const QFont & *font*, int *flags*, const QString & *text*) const [virtual]

Returns the size, that is needed to render text

Parameters

font Font of the text

flags Bitwise OR of the flags used like in QPainter::drawText

text Text to be rendered

Returns

Caluclated size

Implements QwtTextEngine.

12.38 QwtMetricsMap Class Reference

A Map to translate between layout, screen and paint device metrics.

`#include <qwt_layout_metrics.h>`

Public Member Functions

- bool **isIdentity** () const
- void **setMetrics** (const QPaintDevice *layoutMetrics, const QPaintDevice *deviceMetrics)
- int **layoutToDeviceX** (int x) const
- int **deviceToLayoutX** (int x) const
- int **screenToLayoutX** (int x) const
- int **layoutToScreenX** (int x) const
- int **layoutToDeviceY** (int y) const
- int **deviceToLayoutY** (int y) const
- int **screenToLayoutY** (int y) const
- int **layoutToScreenY** (int y) const
- QPoint **layoutToDevice** (const QPoint &, const QPainter *=NULL) const
- QPoint **deviceToLayout** (const QPoint &, const QPainter *=NULL) const
- QPoint **screenToLayout** (const QPoint &) const
- QPoint **layoutToScreen** (const QPoint &point) const
- QSize **layoutToDevice** (const QSize &) const
- QSize **deviceToLayout** (const QSize &) const
- QSize **screenToLayout** (const QSize &) const
- QSize **layoutToScreen** (const QSize &) const
- QRect **layoutToDevice** (const QRect &, const QPainter *=NULL) const
- QRect **deviceToLayout** (const QRect &, const QPainter *=NULL) const
- QRect **screenToLayout** (const QRect &) const
- QRect **layoutToScreen** (const QRect &) const
- QwtPolygon **layoutToDevice** (const QwtPolygon &, const QPainter *=NULL) const
- QwtPolygon **deviceToLayout** (const QwtPolygon &, const QPainter *=NULL) const

Static Public Member Functions

- static QwtPolygon translate (const QMatrix &, const QwtPolygon &)
- static QRect translate (const QMatrix &, const QRect &)

12.38.1 Detailed Description

A Map to translate between layout, screen and paint device metrics. Qt3 supports painting in integer coordinates only. Therefore it is not possible to scale the layout in screen coordinates to layouts in higher resolutions (f.e printing) without losing the higher precision. QwtMetricsMap is used to incorporate the various widget attributes (always in screen resolution) into the layout/printing code of QwtPlot.

Qt4 is able to paint floating point based coordinates, what makes it possible always to render in screen coordinates (with a common scale factor). QwtMetricsMap will be obsolete as soon as Qt3 support has been dropped (Qwt 6.x).

12.38.2 Member Function Documentation

12.38.2.1 QRect QwtMetricsMap::translate (const QMatrix & *m*, const QRect & *rect*) [static]

Wrapper for QMatrix::mapRect.

Parameters

m Matrix

rect Rectangle to translate

Returns

Translated rectangle

12.38.2.2 QwtPolygon QwtMetricsMap::translate (const QMatrix & *m*, const QwtPolygon & *pa*) [static]

Wrapper for QMatrix::map.

Parameters

m Matrix

pa Polygon to translate

Returns

Translated polygon

12.39 QwtPainter Class Reference

A collection of QPainter workarounds.

`#include <qwt_painter.h>`

Static Public Member Functions

- static void setMetricsMap (const QPaintDevice *layout, const QPaintDevice *device)
- static void setMetricsMap (const QwtMetricsMap &)
- static void resetMetricsMap ()
- static const QwtMetricsMap & metricsMap ()
- static void setDeviceClipping (bool)
- static bool deviceClipping ()
- static const QRect & deviceClipRect ()
- static void setClipRect (QPainter *, const QRect &)
- static void drawText (QPainter *, int x, int y, const QString &)
- static void drawText (QPainter *, const QPoint &, const QString &)
- static void drawText (QPainter *, int x, int y, int w, int h, int flags, const QString &)
- static void drawText (QPainter *, const QRect &, int flags, const QString &)
- static void drawSimpleRichText (QPainter *, const QRect &, int flags, QTextDocument &)

- static void drawRect (QPainter ∗, int x, int y, int w, int h)
- static void drawRect (QPainter ∗, const QRect &rect)
- static void fillRect (QPainter ∗, const QRect &, const QBrush &)
- static void drawEllipse (QPainter ∗, const QRect &)
- static void drawPie (QPainter ∗, const QRect &r, int a, int alen)
- static void drawLine (QPainter ∗, int x1, int y1, int x2, int y2)
- static void drawLine (QPainter ∗, const QPoint &p1, const QPoint &p2)
- static void drawPolygon (QPainter ∗, const QwtPolygon &pa)
- static void drawPolyline (QPainter ∗, const QwtPolygon &pa)
- static void drawPoint (QPainter ∗, int x, int y)
- static void drawRoundFrame (QPainter ∗, const QRect &, int width, const QPalette &, bool sunken)
- static void **drawFocusRect** (QPainter ∗, QWidget ∗)
- static void **drawFocusRect** (QPainter ∗, QWidget ∗, const QRect &)
- static void **drawColorBar** (QPainter ∗painter, const QwtColorMap &, const QwtDoubleInterval &, const QwtScaleMap &, Qt::Orientation, const QRect &)
- static QPen scaledPen (const QPen &)

12.39.1 Detailed Description

A collection of QPainter workarounds. 1) Clipping to coordinate system limits (Qt3 only)

On X11 pixel coordinates are stored in shorts. Qt produces overruns when mapping QCOORDS to shorts.

2) Scaling to device metrics

QPainter scales fonts, line and fill patterns to the metrics of the paint device. Other values like the geometries of rects, points remain device independend. To enable a device independent widget implementation, QwtPainter adds scaling of these geometries. (Unfortunately QPainter::scale scales both types of paintings, so the objects of the first type would be scaled twice).

12.39.2 Member Function Documentation

12.39.2.1 bool QwtPainter::deviceClipping () [inline, static]

Returns whether device clipping is enabled. On X11 the default is enabled, otherwise it is disabled.

See also

QwtPainter::setDeviceClipping()

12.39.2.2 const QRect & QwtPainter::deviceClipRect () [static]

Returns rect for device clipping

See also

QwtPainter::setDeviceClipping()

12.39.2.3 void QwtPainter::drawEllipse (QPainter ∗ *painter*, const QRect & *rect*) [static]

Wrapper for QPainter::drawEllipse()

12.39.2.4 void QwtPainter::drawLine (QPainter * *painter*, const QPoint & *p1*, const QPoint & *p2*) [inline, static]

Wrapper for QPainter::drawLine().

12.39.2.5 void QwtPainter::drawLine (QPainter * *painter*, int *x1*, int *y1*, int *x2*, int *y2*) [static]

Wrapper for QPainter::drawLine()

12.39.2.6 void QwtPainter::drawPie (QPainter * *painter*, const QRect & *rect*, int *a*, int *alen*) [static]

Wrapper for QPainter::drawPie()

12.39.2.7 void QwtPainter::drawPoint (QPainter * *painter*, int *x*, int *y*) [static]

Wrapper for QPainter::drawPoint()

12.39.2.8 void QwtPainter::drawPolygon (QPainter * *painter*, const QwtPolygon & *pa*) [static]

Wrapper for QPainter::drawPolygon()

12.39.2.9 void QwtPainter::drawPolyline (QPainter * *painter*, const QwtPolygon & *pa*) [static]

Wrapper for QPainter::drawPolyline()

12.39.2.10 void QwtPainter::drawRect (QPainter * *painter*, const QRect & *rect*) [static]

Wrapper for QPainter::drawRect()

12.39.2.11 void QwtPainter::drawRect (QPainter * *painter*, int *x*, int *y*, int *w*, int *h*) [static]

Wrapper for QPainter::drawRect()

12.39.2.12 void QwtPainter::drawRoundFrame (QPainter * *painter*, const QRect & *rect*, int *width*, const QPalette & *palette*, bool *sunken*) [static]

Draw a round frame.

12.39.2.13 void QwtPainter::drawSimpleRichText (QPainter * *painter*, const QRect & *rect*, int *flags*, QTextDocument & *text*) [static]

Wrapper for QSimpleRichText::draw()

12.39.2.14 void QwtPainter::drawText (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.15 void QwtPainter::drawText (QPainter * *painter*, int *x*, int *y*, int *w*, int *h*, int *flags*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.16 void QwtPainter::drawText (QPainter * *painter*, const QPoint & *pos*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.17 void QwtPainter::drawText (QPainter * *painter*, int *x*, int *y*, const QString & *text*) [static]

Wrapper for QPainter::drawText()

12.39.2.18 void QwtPainter::fillRect (QPainter * *painter*, const QRect & *rect*, const QBrush & *brush*) [static]

Wrapper for QPainter::fillRect()

12.39.2.19 const QwtMetricsMap & QwtPainter::metricsMap () [static]

Returns

Metrics map

12.39.2.20 void QwtPainter::resetMetricsMap () [static]

Reset the metrics map to the ratio 1:1

See also

QwtPainter::setMetricsMap(), QwtPainter::resetMetricsMap()

12.39.2.21 QPen QwtPainter::scaledPen (const QPen & *pen*) [static]

Scale a pen according to the layout metrics.

The width of non cosmetic pens is scaled from screen to layout metrics, so that they look similar on paint devices with different resolutions.

Parameters

pen Unscaled pen

Returns

Scaled pen

12.39.2.22 void QwtPainter::setClipRect (QPainter * *painter*, const QRect & *rect*) [static]

Wrapper for QPainter::setClipRect()

12.39.2.23 void QwtPainter::setDeviceClipping (bool *enable*) [static]

En/Disable device clipping.

On X11 the default for device clipping is enabled, otherwise it is disabled.

See also

QwtPainter::deviceClipping()

12.39.2.24 void QwtPainter::setMetricsMap (const QwtMetricsMap & *map*) [static]

Change the metrics map

See also

QwtPainter::resetMetricsMap(), QwtPainter::metricsMap()

12.39.2.25 void QwtPainter::setMetricsMap (const QPaintDevice * *layout*, const QPaintDevice * *device*) [static]

Scale all QwtPainter drawing operations using the ratio QwtPaintMetrics(from).logicalDpiX() / QwtPaintMetrics(to).logicalDpiX() and QwtPaintMetrics(from).logicalDpiY() / QwtPaintMetrics(to).logicalDpiY()

See also

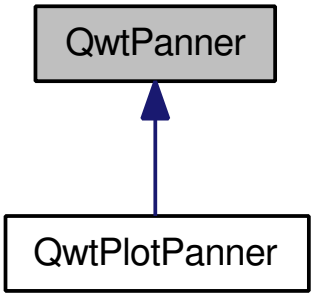
QwtPainter::resetScaleMetrics(), QwtPainter::scaleMetricsX(), QwtPainter::scaleMetricsY()

12.40 QwtPanner Class Reference

QwtPanner provides panning of a widget.

`#include <qwt_panner.h>`

Inheritance diagram for QwtPanner:



Signals

- void panned (int dx, int dy)
- void moved (int dx, int dy)

Public Member Functions

- QwtPanner (QWidget *parent)
- virtual ~QwtPanner ()
- void setEnabled (bool)
- bool isEnabled () const
- void setMouseButton (int button, int buttonState=Qt::NoButton)
- void getMouseButton (int &button, int &buttonState) const
- void setAbortKey (int key, int state=Qt::NoButton)
- void getAbortKey (int &key, int &state) const
- void setCursor (const QCursor &)
- const QCursor cursor () const
- void setOrientations (Qt::Orientations)
- Qt::Orientations orientations () const
- bool isOrientationEnabled (Qt::Orientation) const
- virtual bool eventFilter (QObject *, QEvent *)

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 *)

12.40.1 Detailed Description

QwtPanner provides panning of a widget. QwtPanner grabs the contents of a widget, that can be dragged in all directions. The offset between the start and the end position is emitted by the panned signal.

QwtPanner grabs the content of the widget into a pixmap and moves the pixmap around, without initiating any repaint events for the widget. Areas, that are not part of content are not painted while panning in in process. This makes panning fast enough for widgets, where repaints are too slow for mouse movements.

For widgets, where repaints are very fast it might be better to implement panning manually by mapping mouse events into paint events.

12.40.2 Constructor & Destructor Documentation

12.40.2.1 QwtPanner::QwtPanner (QWidget * *parent*)

Creates an panner that is enabled for the left mouse button.

Parameters

parent Parent widget to be panned

12.40.2.2 QwtPanner::~QwtPanner () `[virtual]`

Destructor.

12.40.3 Member Function Documentation

12.40.3.1 const QCursor QwtPanner::cursor () const

Returns

Cursor that is active while panning

See also

setCursor()

12.40.3.2 bool QwtPanner::eventFilter (QObject * *o*, QEvent * *e*) `[virtual]`

Event filter.

When isEnabled() the mouse events of the observed widget are filtered.

See also

widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseMoveEvent()

12.40.3.3 void QwtPanner::getAbortKey (int & *key*, int & *state*) const

Get the abort key.

12.40.3.4 void QwtPanner::getMouseButton (int & *button*, int & *buttonState*) const

Get the mouse button.

12.40.3.5 bool QwtPanner::isEnabled () const

Returns

true when enabled, false otherwise

See also

setEnabled, eventFilter()

12.40.3.6 bool QwtPanner::isOrientationEnabled (Qt::Orientation *o*) const

Return true if a orientatio is enabled

See also

orientations(), setOrientations()

12.40.3.7 void QwtPanner::moved (int *dx*, int *dy*) `[signal]`

Signal emitted, while the widget moved, but panning is not finished.

Parameters

dx Offset in horizontal direction

dy Offset in vertical direction

12.40.3.8 Qt::Orientations QwtPanner::orientations () const

Return the orientation, where paning is enabled.

12.40.3.9 void QwtPanner::paintEvent (QPaintEvent * *pe*) [protected, virtual]

Paint event.

Repaint the grabbed pixmap on its current position and fill the empty spaces by the background of the parent widget.

Parameters

pe Paint event

12.40.3.10 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

12.40.3.11 void QwtPanner::setAbortKey (int *key*, int *state* = Qt::NoButton)

Change the abort key The defaults are Qt::Key_Escape and Qt::NoButton

Parameters

key Key (See Qt::Keycode)

state State

12.40.3.12 void QwtPanner::setCursor (const QCursor & *cursor*)

Change the cursor, that is active while panning The default is the cursor of the parent widget.

Parameters

cursor New cursor

See also

setCursor()

12.40.3.13 void QwtPanner::setEnabled (bool *on*)

En/disable the panner.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

on true or false

See also

isEnabled(), eventFilter()

12.40.3.14 void QwtPanner::setMouseButton (int *button*, int *buttonState* = `Qt::NoButton`)

Change the mouse button The defaults are Qt::LeftButton and Qt::NoButton

12.40.3.15 void QwtPanner::setOrientations (Qt::Orientations *o*)

Set the orientations, where panning is enabled The default value is in both directions: Qt::Horizontal | Qt::Vertical

/param o Orientation

12.40.3.16 void QwtPanner::widgetKeyPressEvent (QKeyEvent * *ke*) `[protected, virtual]`

Handle a key press event for the observed widget.

Parameters

ke Key event

See also

eventFilter(), widgetKeyReleaseEvent()

12.40.3.17 void QwtPanner::widgetKeyReleaseEvent (QKeyEvent *) `[protected, virtual]`

Handle a key release event for the observed widget.

See also

eventFilter(), widgetKeyReleaseEvent()

12.40.3.18 void QwtPanner::widgetMouseMoveEvent (QMouseEvent * *me*) `[protected, virtual]`

Handle a mouse move event for the observed widget.

Parameters

me Mouse event

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent()

12.40.3.19 void QwtPanner::widgetMousePressEvent (QMouseEvent * *me*) [protected, virtual]

Handle a mouse press event for the observed widget.

Parameters

me Mouse event

See also

eventFilter(), widgetMouseReleaseEvent(), widgetMouseMoveEvent(),

12.40.3.20 void QwtPanner::widgetMouseReleaseEvent (QMouseEvent * *me*) [protected, virtual]

Handle a mouse release event for the observed widget.

Parameters

me Mouse event

See also

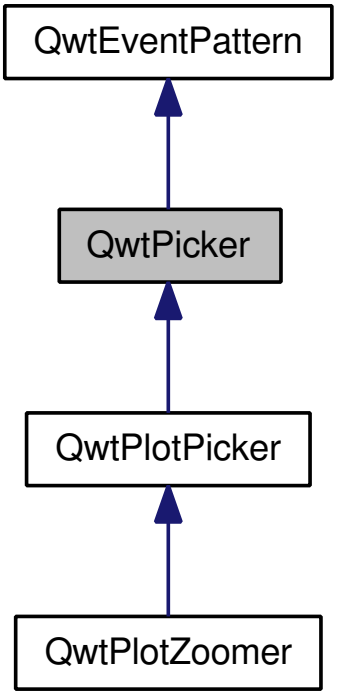
eventFilter(), widgetMousePressEvent(), widgetMouseMoveEvent(),

12.41 QwtPicker Class Reference

QwtPicker provides selections on a widget.

```
#include <qwt_picker.h>
```

Inheritance diagram for QwtPicker:



Public Types

- enum SelectionType {
  NoSelection = 0,
  PointSelection = 1,
  RectSelection = 2,
  PolygonSelection = 4 }
- enum RectSelectionType {
  CornerToCorner = 64,
  CenterToCorner = 128,
  CenterToRadius = 256 }
- enum SelectionMode {
  ClickSelection = 1024,
  DragSelection = 2048 }
- enum RubberBand {
  NoRubberBand = 0,
  HLineRubberBand,
  VLineRubberBand,
  CrossRubberBand,
  RectRubberBand,
  EllipseRubberBand,

PolygonRubberBand,

UserRubberBand = 100 }

- enum DisplayMode {

  AlwaysOff,

  AlwaysOn,

  ActiveOnly }
- enum ResizeMode {

  Stretch,

  KeepSize }

Signals

- void selected (const QwtPolygon &pa)
- void appended (const QPoint &pos)
- void moved (const QPoint &pos)
- void changed (const QwtPolygon &pa)

Public Member Functions

- QwtPicker (QWidget *parent)
- QwtPicker (int selectionFlags, RubberBand rubberBand, DisplayMode trackerMode, QWidget *)
- virtual ~QwtPicker ()
- virtual void setSelectionFlags (int)
- int selectionFlags () const
- virtual void setRubberBand (RubberBand)
- RubberBand rubberBand () const
- virtual void setTrackerMode (DisplayMode)
- DisplayMode trackerMode () const
- virtual void setResizeMode (ResizeMode)
- ResizeMode resizeMode () const
- virtual void setRubberBandPen (const QPen &)
- QPen rubberBandPen () const
- virtual void setTrackerPen (const QPen &)
- QPen trackerPen () const
- virtual void setTrackerFont (const QFont &)
- QFont trackerFont () const
- bool isEnabled () const
- virtual void setEnabled (bool)
- bool isActive () const
- virtual bool eventFilter (QObject *, QEvent *)
- QWidget * parentWidget ()
- const QWidget * parentWidget () const
- virtual QRect pickRect () const
- const QwtPolygon & selection () const
- virtual void drawRubberBand (QPainter *) const
- virtual void drawTracker (QPainter *) const
- virtual QwtText trackerText (const QPoint &pos) const
- QPoint trackerPosition () const
- QRect trackerRect (const QFont &) const

Protected Member Functions

- virtual bool accept (QwtPolygon &selection) const
- virtual void transition (const QEvent *)
- virtual void begin ()
- virtual void append (const QPoint &)
- virtual void move (const QPoint &)
- virtual bool end (bool ok=true)
- 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 widgetLeaveEvent (QEvent *)
- virtual void stretchSelection (const QSize &oldSize, const QSize &newSize)
- virtual QwtPickerMachine * stateMachine (int) const
- virtual void updateDisplay ()
- const QWidget * rubberBandWidget () const
- const QWidget * trackerWidget () const

12.41.1 Detailed Description

QwtPicker provides selections on a widget. QwtPicker filters all mouse and keyboard events of a widget and translates them into an array of selected points. Depending on the QwtPicker::SelectionType the selection might be a single point, a rectangle or a polygon. The selection process is supported by optional rubberbands (rubberband selection) and position trackers.

QwtPicker is useful for widgets where the event handlers can't be overloaded, like for components of composite widgets. It offers alternative handlers for mouse and key events.

Example

```
#include <qwt_picker.h>

QwtPicker *picker = new QwtPicker(widget);
picker->setTrackerMode(QwtPicker::ActiveOnly);
connect(picker, SIGNAL(selected(const QwtPolygon &)), ...);

// emit the position of clicks on widget
picker->setSelectionFlags(QwtPicker::PointSelection | QwtPicker::ClickSelection);

    ...

// now select rectangles
picker->setSelectionFlags(QwtPicker::RectSelection | QwtPicker::DragSelection);
picker->setRubberBand(QwtPicker::RectRubberBand);
```

The selection process uses the commands begin(), append(), move() and end(). append() adds a new point to the selection, move() changes the position of the latest point.

The commands are initiated from a small state machine (QwtPickerMachine) that translates mouse and key events. There are a couple of predefined state machines for point, rect and polygon selections. The

selectionFlags() control which one should be used. It is possible to use other machines by overloading stateMachine().

The picker is active (isActive()), between begin() and end(). In active state the rubberband is displayed, and the tracker is visible in case of trackerMode is ActiveOnly or AlwaysOn.

The cursor can be moved using the arrow keys. All selections can be aborted using the abort key. (QwtEventPattern::KeyPatternCode)

Warning

In case of QWidget::NoFocus the focus policy of the observed widget is set to QWidget::WheelFocus and mouse tracking will be manipulated for ClickSelection while the picker is active, or if trackerMode() is AlwayOn.

12.41.2 Member Enumeration Documentation

12.41.2.1 enum QwtPicker::DisplayMode

- AlwaysOff
  Display never.
- AlwaysOn
  Display always.
- ActiveOnly
  Display only when the selection is active.

See also

QwtPicker::setTrackerMode(), QwtPicker::trackerMode(), QwtPicker::isActive()

12.41.2.2 enum QwtPicker::RectSelectionType

Selection subtype for RectSelection This enum type describes the type of rectangle selections. It can be or'd with QwtPicker::RectSelectionType and QwtPicker::SelectionMode and passed to QwtPicker::setSelectionFlags().

- CornerToCorner
  The first and the second selected point are the corners of the rectangle.
- CenterToCorner
  The first point is the center, the second a corner of the rectangle.
- CenterToRadius
  The first point is the center of a quadrat, calculated by the maximum of the x- and y-distance.

The default value is CornerToCorner.

See also

QwtPicker::setSelectionFlags(), QwtPicker::selectionFlags()

12.41.2.3 enum QwtPicker::ResizeMode

Controls what to do with the selected points of an active selection when the observed widget is resized.

- Stretch
  All points are scaled according to the new size,
- KeepSize
  All points remain unchanged.

The default value is Stretch.

See also

QwtPicker::setResizeMode(), QwtPicker::resize()

12.41.2.4 enum QwtPicker::RubberBand

Rubberband style

- NoRubberBand
  No rubberband.
- HLineRubberBand & PointSelection
  A horizontal line.
- VLineRubberBand & PointSelection
  A vertical line.
- CrossRubberBand & PointSelection
  A horizontal and a vertical line.
- RectRubberBand & RectSelection
  A rectangle.
- EllipseRubberBand & RectSelection
  An ellipse.
- PolygonRubberBand &PolygonSelection
  A polygon.
- UserRubberBand
  Values >= UserRubberBand can be used to define additional rubber bands.

The default value is NoRubberBand.

See also

QwtPicker::setRubberBand(), QwtPicker::rubberBand()

12.41.2.5 enum QwtPicker::SelectionMode

Values of this enum type or'd together with a SelectionType value identifies which state machine should be used for the selection.

The default value is ClickSelection.

See also

stateMachine()

12.41.2.6 enum QwtPicker::SelectionType

This enum type describes the type of a selection. It can be or'd with QwtPicker::RectSelectionType and QwtPicker::SelectionMode and passed to QwtPicker::setSelectionFlags()

- NoSelection

  Selection is disabled. Note this is different to the disabled state, as you might have a tracker.
- PointSelection

  Select a single point.
- RectSelection

  Select a rectangle.
- PolygonSelection

  Select a polygon.

The default value is NoSelection.

See also

QwtPicker::setSelectionFlags(), QwtPicker::selectionFlags()

12.41.3 Constructor & Destructor Documentation

12.41.3.1 QwtPicker::QwtPicker (QWidget * *parent*) [explicit]

Constructor

Creates an picker that is enabled, but where selection flag is set to NoSelection, rubberband and tracker are disabled.

Parameters

parent Parent widget, that will be observed

12.41.3.2 QwtPicker::QwtPicker (int *selectionFlags*, RubberBand *rubberBand*, DisplayMode *trackerMode*, QWidget * *parent*) [explicit]

Constructor

Parameters

selectionFlags Or'd value of SelectionType, RectSelectionType and SelectionMode

rubberBand Rubberband style

trackerMode Tracker mode

parent Parent widget, that will be observed

12.41.3.3 QwtPicker::~QwtPicker () [virtual]

Destructor.

12.41.4 Member Function Documentation

12.41.4.1 bool QwtPicker::accept (QwtPolygon & *selection*) const [protected, virtual]

Validate and fixup the selection.

Accepts all selections unmodified

Parameters

selection Selection to validate and fixup

Returns

true, when accepted, false otherwise

Reimplemented in QwtPlotZoomer.

12.41.4.2 void QwtPicker::append (const QPoint & *pos*) [protected, virtual]

Append a point to the selection and update rubberband and tracker. The appended() signal is emitted.

Parameters

pos Additional point

See also

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

Reimplemented in QwtPlotPicker.

12.41.4.3 void QwtPicker::appended (const QPoint & *pos*) [signal]

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

Parameters

pos Position of the appended point.

See also

append(). moved()

12.41.4.4 void QwtPicker::begin () [protected, virtual]

Open a selection setting the state to active

See also

isActive(), end(), append(), move()

Reimplemented in QwtPlotZoomer.

12.41.4.5 void QwtPicker::changed (const QwtPolygon & *pa*) [signal]

A signal emitted when the active selection has been changed. This might happen when the observed widget is resized.

Parameters

pa Changed selection

See also

stretchSelection()

12.41.4.6 void QwtPicker::drawRubberBand (QPainter * *painter*) const [virtual]

Draw a rubberband , depending on rubberBand() and selectionFlags()

Parameters

painter Painter, initialized with clip rect

See also

rubberBand(), RubberBand, selectionFlags()

12.41.4.7 void QwtPicker::drawTracker (QPainter * *painter*) const [virtual]

Draw the tracker

Parameters

painter Painter

See also

trackerRect(), trackerText()

12.41.4.8 bool QwtPicker::end (bool *ok* = `true`) `[protected, virtual]`

Close a selection setting the state to inactive.

The selection is validated and maybe fixed by QwtPicker::accept().

Parameters

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 QwtPlotPicker, and QwtPlotZoomer.

12.41.4.9 bool QwtPicker::eventFilter (QObject * *o*, QEvent * *e*) `[virtual]`

Event filter.

When isEnabled() == true all events of the observed widget are filtered. Mouse and keyboard events are translated into widgetMouse- and widgetKey- and widgetWheel-events. Paint and Resize events are handled to keep rubberband and tracker up to date.

See also

event(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

12.41.4.10 bool QwtPicker::isActive () const

A picker is active between begin() and end().

Returns

true if the selection is active.

12.41.4.11 bool QwtPicker::isEnabled () const

Returns

true when enabled, false otherwise

See also

setEnabled(), eventFilter()

12.41.4.12 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 QwtPlotPicker.

12.41.4.13 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()

12.41.4.14 const QWidget * QwtPicker::parentWidget () const

Return the parent widget, where the selection happens.

12.41.4.15 QWidget * QwtPicker::parentWidget ()

Return the parent widget, where the selection happens.

12.41.4.16 QRect QwtPicker::pickRect () const [virtual]

Find the area of the observed widget, where selection might happen.

Returns

QFrame::contentsRect() if it is a QFrame, QWidget::rect() otherwise.

12.41.4.17 void QwtPicker::reset () [protected, virtual]

Reset the state machine and terminate (end(false)) the selection

12.41.4.18 QwtPicker::ResizeMode QwtPicker::resizeMode () const

Returns

Resize mode

See also

setResizeMode(), ResizeMode

12.41.4.19 QwtPicker::RubberBand QwtPicker::rubberBand () const

Returns

Rubberband style

See also

setRubberBand(), RubberBand, rubberBandPen()

12.41.4.20 QPen QwtPicker::rubberBandPen () const

Returns

Rubberband pen

See also

setRubberBandPen(), rubberBand()

12.41.4.21 const QWidget * QwtPicker::rubberBandWidget () const [protected]

Returns

Widget displaying the rubberband

12.41.4.22 void QwtPicker::selected (const QwtPolygon & *pa*) [signal]

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

Parameters

pa Selected points

12.41.4.23 const QwtPolygon & QwtPicker::selection () const

Return Selected points.

12.41.4.24 int QwtPicker::selectionFlags () const

Returns

Selection flags, an Or'd value of SelectionType, RectSelectionType and SelectionMode.

See also

setSelectionFlags(), SelectionType, RectSelectionType, SelectionMode

12.41.4.25 void QwtPicker::setEnabled (bool *enabled*) [virtual]

En/disable the picker.

When enabled is true an event filter is installed for the observed widget, otherwise the event filter is removed.

Parameters

enabled true or false

See also

isEnabled(), eventFilter()

12.41.4.26 void QwtPicker::setResizeMode (ResizeMode *mode*) [virtual]

Set the resize mode.

The resize mode controls what to do with the selected points of an active selection when the observed widget is resized.

Stretch means the points are scaled according to the new size, KeepSize means the points remain unchanged.

The default mode is Stretch.

Parameters

mode Resize mode

See also

resizeMode(), ResizeMode

12.41.4.27 void QwtPicker::setRubberBand (RubberBand *rubberBand*) [virtual]

Set the rubberband style

Parameters

rubberBand Rubberband style The default value is NoRubberBand.

See also

rubberBand(), RubberBand, setRubberBandPen()

12.41.4.28 void QwtPicker::setRubberBandPen (const QPen & *pen*) [virtual]

Set the pen for the rubberband

Parameters

pen Rubberband pen

See also

rubberBandPen(), setRubberBand()

12.41.4.29 void QwtPicker::setSelectionFlags (int *flags*) [virtual]

Set the selection flags

Parameters

flags Or'd value of SelectionType, RectSelectionType and SelectionMode. The default value is NoSelection.

See also

selectionFlags(), SelectionType, RectSelectionType, SelectionMode

Reimplemented in QwtPlotZoomer.

12.41.4.30 void QwtPicker::setTrackerFont (const QFont & *font*) [virtual]

Set the font for the tracker

Parameters

font Tracker font

See also

trackerFont(), setTrackerMode(), setTrackerPen()

12.41.4.31 void QwtPicker::setTrackerMode (DisplayMode *mode*) [virtual]

Set the display mode of the tracker.

A tracker displays information about current position of the cursor as a string. The display mode controls if the tracker has to be displayed whenever the observed widget has focus and cursor (AlwaysOn), never (AlwaysOff), or only when the selection is active (ActiveOnly).

Parameters

mode Tracker display mode

Warning

In case of AlwaysOn, mouseTracking will be enabled for the observed widget.

See also

trackerMode(), DisplayMode

12.41.4.32 void QwtPicker::setTrackerPen (const QPen & *pen*) [virtual]

Set the pen for the tracker

Parameters

pen Tracker pen

See also

trackerPen(), setTrackerMode(), setTrackerFont()

12.41.4.33 QwtPickerMachine * QwtPicker::stateMachine (int *flags*) const [protected, virtual]

Create a state machine depending on the selection flags.

- PointSelection | ClickSelection
  QwtPickerClickPointMachine()
- PointSelection | DragSelection
  QwtPickerDragPointMachine()
- RectSelection | ClickSelection
  QwtPickerClickRectMachine()
- RectSelection | DragSelection
  QwtPickerDragRectMachine()
- PolygonSelection
  QwtPickerPolygonMachine()

See also

setSelectionFlags()

12.41.4.34 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()

12.41.4.35 QFont QwtPicker::trackerFont () const

Returns

Tracker font

See also

setTrackerFont(), trackerMode(), trackerPen()

12.41.4.36 QwtPicker::DisplayMode QwtPicker::trackerMode () const

Returns

Tracker display mode

See also

setTrackerMode(), DisplayMode

12.41.4.37 QPen QwtPicker::trackerPen () const

Returns

Tracker pen

See also

setTrackerPen(), trackerMode(), trackerFont()

12.41.4.38 QPoint QwtPicker::trackerPosition () const

Returns

Current position of the tracker

12.41.4.39 QRect QwtPicker::trackerRect (const QFont & *font*) const

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()

12.41.4.40 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 QwtPlotPicker.

12.41.4.41 const QWidget * QwtPicker::trackerWidget () const [protected]

Returns

Widget displaying the tracker text

12.41.4.42 void QwtPicker::transition (const QEvent * *e*) [protected, virtual]

Passes an event to the state machine and executes the resulting commands. Append and Move commands use the current position of the cursor (QCursor::pos()).

Parameters

e Event

12.41.4.43 void QwtPicker::updateDisplay () [protected, virtual]

Update the state of rubberband and tracker label.

12.41.4.44 void QwtPicker::widgetKeyPressEvent (QKeyEvent * *ke*) [protected, virtual]

Handle a key press event for the observed widget.

Selections can be completely done by the keyboard. The arrow keys move the cursor, the abort key aborts a selection. All other keys are handled by the current state machine.

See also

QwtPicker, selectionFlags()
eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyReleaseEvent(), stateMachine(), QwtEventPattern::KeyPatternCode

Reimplemented in QwtPlotZoomer.

12.41.4.45 void QwtPicker::widgetKeyReleaseEvent (QKeyEvent * *ke*) [protected, virtual]

Handle a key release event for the observed widget.

Passes the event to the state machine.

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), stateMachine()

12.41.4.46 void QwtPicker::widgetLeaveEvent (QEvent *) [protected, virtual]

Handle a leave event for the observed widget.

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

12.41.4.47 void QwtPicker::widgetMouseDoubleClickEvent (QMouseEvent * *me*) [protected, virtual]

Handle mouse double click event for the observed widget.

Empty implementation, does nothing.

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseMoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

12.41.4.48 void QwtPicker::widgetMouseMoveEvent (QMouseEvent * *e*) [protected, virtual]

Handle a mouse move event for the observed widget.

Move the last point of the selection in case of isActive() == true

See also

eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClick-Event(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

12.41.4.49 void QwtPicker::widgetMousePressEvent (QMouseEvent * *e*) [protected, virtual]

Handle a mouse press event for the observed widget.

Begin and/or end a selection depending on the selection flags.

See also

QwtPicker, selectionFlags()
eventFilter(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouse-MoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

12.41.4.50 void QwtPicker::widgetMouseReleaseEvent (QMouseEvent * *e*) [protected, virtual]

Handle a mouse relase event for the observed widget.

End a selection depending on the selection flags.

See also

QwtPicker, selectionFlags()
eventFilter(), widgetMousePressEvent(), widgetMouseDoubleClickEvent(), widgetMouse-MoveEvent(), widgetWheelEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

Reimplemented in QwtPlotZoomer.

12.41.4.51 void QwtPicker::widgetWheelEvent (QWheelEvent * *e*) [protected, virtual]

Handle a wheel event for the observed widget.

Move the last point of the selection in case of isActive() == true

See also

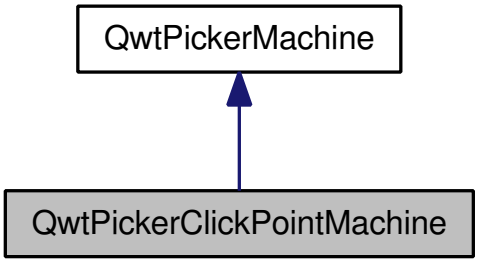
eventFilter(), widgetMousePressEvent(), widgetMouseReleaseEvent(), widgetMouseDoubleClickEvent(), widgetMouseMoveEvent(), widgetKeyPressEvent(), widgetKeyReleaseEvent()

12.42 QwtPickerClickPointMachine Class Reference

A state machine for point selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerClickPointMachine:



Public Member Functions

- virtual CommandList transition (const QwtEventPattern &, const QEvent *)

12.42.1 Detailed Description

A state machine for point selections. Pressing QwtEventPattern::MouseSelect1 or QwtEventPattern::KeySelect1 selects a point.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

12.42.2 Member Function Documentation

12.42.2.1 QwtPickerMachine::CommandList QwtPickerClickPointMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

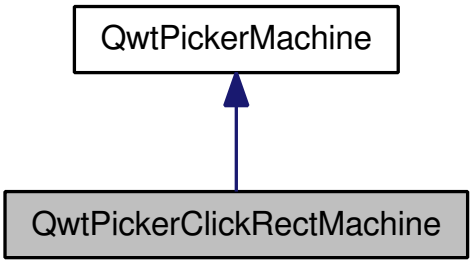
Implements QwtPickerMachine.

12.43 QwtPickerClickRectMachine Class Reference

A state machine for rectangle selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerClickRectMachine:



Public Member Functions

- virtual CommandList transition (const QwtEventPattern &, const QEvent *)

12.43.1 Detailed Description

A state machine for rectangle selections. Pressing QwtEventPattern::MouseSelect1 starts the selection, releasing it selects the first point. Pressing it again selects the second point and terminates the selection. Pressing QwtEventPattern::KeySelect1 also starts the selection, a second press selects the first point. A third one selects the second point and terminates the selection.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

12.43.2 Member Function Documentation

12.43.2.1 QwtPickerMachine::CommandList QwtPickerClickRectMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

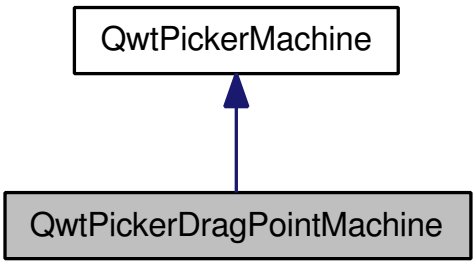
Implements QwtPickerMachine.

12.44 QwtPickerDragPointMachine Class Reference

A state machine for point selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerDragPointMachine:



Public Member Functions

- virtual CommandList transition (const QwtEventPattern &, const QEvent *)

12.44.1 Detailed Description

A state machine for point selections. Pressing QwtEventPattern::MouseSelect1 or QwtEventPattern::KeySelect1 starts the selection, releasing QwtEventPattern::MouseSelect1 or a second press of QwtEventPattern::KeySelect1 terminates it.

12.44.2 Member Function Documentation

12.44.2.1 QwtPickerMachine::CommandList QwtPickerDragPointMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

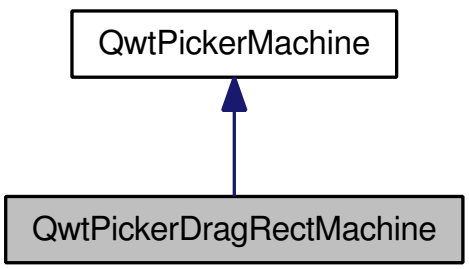
Implements QwtPickerMachine.

12.45 QwtPickerDragRectMachine Class Reference

A state machine for rectangle selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerDragRectMachine:



Public Member Functions

- virtual CommandList transition (const QwtEventPattern &, const QEvent *)

12.45.1 Detailed Description

A state machine for rectangle selections. Pressing QwtEventPattern::MouseSelect1 selects the first point, releasing it the second point. Pressing QwtEventPattern::KeySelect1 also selects the first point, a second press selects the second point and terminates the selection.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

12.45.2 Member Function Documentation

12.45.2.1 QwtPickerMachine::CommandList QwtPickerDragRectMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

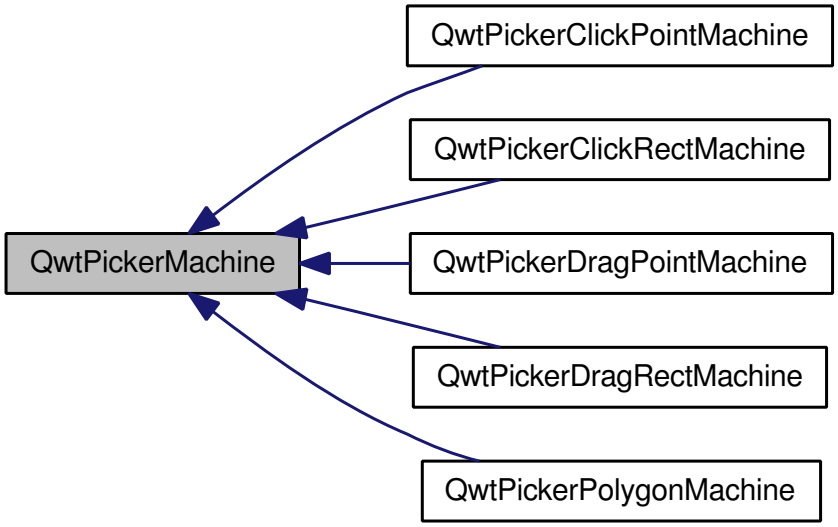
Implements QwtPickerMachine.

12.46 QwtPickerMachine Class Reference

A state machine for QwtPicker selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerMachine:



Public Types

- enum Command {
  Begin,
  Append,
  Move,
  End }
- typedef QList< Command > **CommandList**

Public Member Functions

- virtual ~QwtPickerMachine ()
- virtual CommandList transition (const QwtEventPattern &, const QEvent *)=0
- void reset ()
- int state () const
- void setState (int)

Protected Member Functions

- QwtPickerMachine ()

12.46.1 Detailed Description

A state machine for QwtPicker selections. QwtPickerMachine accepts key and mouse events and translates them into selection commands.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

12.46.2 Member Enumeration Documentation

12.46.2.1 enum QwtPickerMachine::Command

Commands - the output of the state machine.

12.46.3 Constructor & Destructor Documentation

12.46.3.1 QwtPickerMachine::~QwtPickerMachine () [virtual]

Destructor.

12.46.3.2 QwtPickerMachine::QwtPickerMachine () [protected]

Constructor.

12.46.4 Member Function Documentation

12.46.4.1 void QwtPickerMachine::reset ()

Set the current state to 0.

12.46.4.2 void QwtPickerMachine::setState (int *state*)

Change the current state.

12.46.4.3 int QwtPickerMachine::state () const

Return the current state.

12.46.4.4 virtual CommandList QwtPickerMachine::transition (const QwtEventPattern &, const QEvent *) [pure virtual]

Transition.

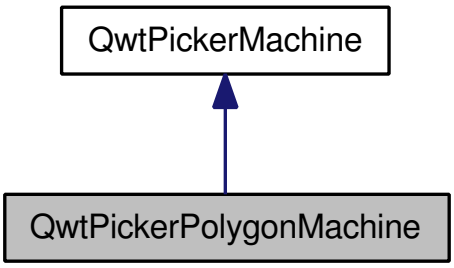
Implemented in QwtPickerClickPointMachine, QwtPickerDragPointMachine, QwtPickerClickRectMachine, QwtPickerDragRectMachine, and QwtPickerPolygonMachine.

12.47 QwtPickerPolygonMachine Class Reference

A state machine for polygon selections.

`#include <qwt_picker_machine.h>`

Inheritance diagram for QwtPickerPolygonMachine:



Public Member Functions

- virtual CommandList transition (const QwtEventPattern &, const QEvent *)

12.47.1 Detailed Description

A state machine for polygon selections. Pressing QwtEventPattern::MouseSelect1 or QwtEventPattern::KeySelect1 starts the selection and selects the first point, or appends a point. Pressing QwtEventPattern::MouseSelect2 or QwtEventPattern::KeySelect2 appends the last point and terminates the selection.

See also

QwtEventPattern::MousePatternCode, QwtEventPattern::KeyPatternCode

12.47.2 Member Function Documentation

12.47.2.1 QwtPickerMachine::CommandList QwtPickerPolygonMachine::transition (const QwtEventPattern & *eventPattern*, const QEvent * *e*) [virtual]

Transition.

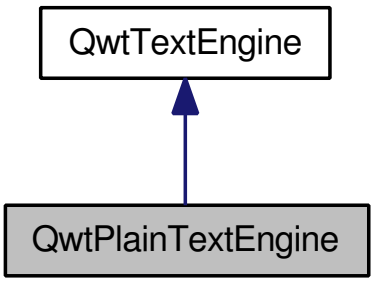
Implements QwtPickerMachine.

12.48 QwtPlainTextEngine Class Reference

A text engine for plain texts.

`#include <qwt_text_engine.h>`

Inheritance diagram for QwtPlainTextEngine:



Public Member Functions

- QwtPlainTextEngine ()
- virtual ~QwtPlainTextEngine ()
- virtual int heightForWidth (const QFont &font, int flags, const QString &text, int width) const
- virtual QSize textSize (const QFont &font, int flags, const QString &text) const
- virtual void draw (QPainter *painter, const QRect &rect, int flags, const QString &text) const
- virtual bool mightRender (const QString &) const
- virtual void textMargins (const QFont &, const QString &, int &left, int &right, int &top, int &bottom) const

12.48.1 Detailed Description

A text engine for plain texts. QwtPlainTextEngine renders texts using the basic Qt classes QPainter and QFontMetrics.

12.48.2 Constructor & Destructor Documentation

12.48.2.1 QwtPlainTextEngine::QwtPlainTextEngine ()

Constructor.

12.48.2.2 QwtPlainTextEngine::~QwtPlainTextEngine () [virtual]

Destructor.

12.48.3 Member Function Documentation

12.48.3.1 void QwtPlainTextEngine::draw (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) const [virtual]

Draw the text in a clipping rectangle.

A wrapper for QPainter::drawText.

Parameters

painter Painter

rect Clipping rectangle

flags Bitwise OR of the flags used like in QPainter::drawText

text Text to be rendered

Implements QwtTextEngine.

12.48.3.2 int QwtPlainTextEngine::heightForWidth (const QFont & *font*, int *flags*, const QString & *text*, int *width*) const [virtual]

Find the height for a given width

Parameters

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.

12.48.3.3 bool QwtPlainTextEngine::mightRender (const QString &) const [virtual]

Test if a string can be rendered by this text engine.

Returns

Always true. All texts can be rendered by QwtPlainTextEngine

Implements QwtTextEngine.

12.48.3.4 void QwtPlainTextEngine::textMargins (const QFont & *font*, const QString &, int & *left*, int & *right*, int & *top*, int & *bottom*) const [virtual]

Return margins around the texts

Parameters

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.

12.48.3.5 QSize QwtPlainTextEngine::textSize (const QFont & *font*, int *flags*, const QString & *text*) const [virtual]

Returns the size, that is needed to render text

Parameters

font Font of the text

flags Bitwise OR of the flags used like in QPainter::drawText

text Text to be rendered

Returns

Caluclated size

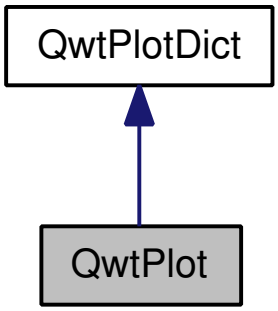
Implements QwtTextEngine.

12.49 QwtPlot Class Reference

A 2-D plotting widget.

`#include <qwt_plot.h>`

Inheritance diagram for QwtPlot:



Public Types

- enum Axis {
  yLeft,
  yRight,
  xBottom,
  xTop,
  axisCnt }
- enum LegendPosition {
  LeftLegend,
  RightLegend,
  BottomLegend,
  TopLegend,
  ExternalLegend }

Public Slots

- virtual void clear ()
- virtual void replot ()
- void autoRefresh ()

Signals

- void legendClicked (QwtPlotItem *plotItem)
- void legendChecked (QwtPlotItem *plotItem, bool on)

Public Member Functions

- QwtPlot (QWidget *=NULL)
- QwtPlot (const QwtText &title, QWidget *p=NULL)
- virtual ~QwtPlot ()
- void applyProperties (const QString &)
- QString grabProperties () const
- void setAutoReplot (bool tf=true)
- bool autoReplot () const
- void print (QPaintDevice &p, const QwtPlotPrintFilter &=QwtPlotPrintFilter()) const
- virtual void print (QPainter *, const QRect &rect, const QwtPlotPrintFilter &=QwtPlotPrintFilter()) const
- QwtPlotLayout * plotLayout ()
- const QwtPlotLayout * plotLayout () const
- void setMargin (int margin)
- int margin () const
- void setTitle (const QString &)
- void setTitle (const QwtText &t)
- QwtText title () const
- QwtTextLabel * titleLabel ()
- const QwtTextLabel * titleLabel () const
- QwtPlotCanvas * canvas ()
- const QwtPlotCanvas * canvas () const
- void setCanvasBackground (const QColor &c)
- const QColor & canvasBackground () const
- void setCanvasLineWidth (int w)
- int canvasLineWidth () const
- virtual QwtScaleMap canvasMap (int axisId) const
- double invTransform (int axisId, int pos) const
- int transform (int axisId, double value) const
- QwtScaleEngine * axisScaleEngine (int axisId)
- const QwtScaleEngine * axisScaleEngine (int axisId) const
- void setAxisScaleEngine (int axisId, QwtScaleEngine *)
- void setAxisAutoScale (int axisId)
- bool axisAutoScale (int axisId) const
- void enableAxis (int axisId, bool tf=true)
- bool axisEnabled (int axisId) const
- void setAxisFont (int axisId, const QFont &f)
- QFont axisFont (int axisId) const
- void setAxisScale (int axisId, double min, double max, double step=0)
- void setAxisScaleDiv (int axisId, const QwtScaleDiv &)
- void setAxisScaleDraw (int axisId, QwtScaleDraw *)
- double axisStepSize (int axisId) const
- const QwtScaleDiv * axisScaleDiv (int axisId) const
- QwtScaleDiv * axisScaleDiv (int axisId)
- const QwtScaleDraw * axisScaleDraw (int axisId) const
- QwtScaleDraw * axisScaleDraw (int axisId)
- const QwtScaleWidget * axisWidget (int axisId) const
- QwtScaleWidget * axisWidget (int axisId)
- void setAxisLabelAlignment (int axisId, Qt::Alignment)

- void setAxisLabelRotation (int axisId, double rotation)
- void setAxisTitle (int axisId, const QString &)
- void setAxisTitle (int axisId, const QwtText &)
- QwtText axisTitle (int axisId) const
- void setAxisMaxMinor (int axisId, int maxMinor)
- int axisMaxMajor (int axisId) const
- void setAxisMaxMajor (int axisId, int maxMajor)
- int axisMaxMinor (int axisId) const
- void insertLegend (QwtLegend *, LegendPosition=QwtPlot::RightLegend, double ratio=-1.0)
- QwtLegend * legend ()
- const QwtLegend * legend () const
- virtual void polish ()
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- virtual void updateLayout ()
- virtual void drawCanvas (QPainter *)
- void updateAxes ()
- virtual bool event (QEvent *)

Protected Slots

- virtual void legendItemClicked ()
- virtual void legendItemChecked (bool)

Protected Member Functions

- virtual void drawItems (QPainter *, const QRect &, const QwtScaleMap maps[axisCnt], const QwtPlotPrintFilter &) const
- virtual void updateTabOrder ()
- virtual void resizeEvent (QResizeEvent *e)
- virtual void printLegendItem (QPainter *, const QWidget *, const QRect &) const
- virtual void printTitle (QPainter *, const QRect &) const
- virtual void printScale (QPainter *, int axisId, int startDist, int endDist, int baseDist, const QRect &) const
- virtual void printCanvas (QPainter *, const QRect &boundingRect, const QRect &canvasRect, const QwtScaleMap maps[axisCnt], const QwtPlotPrintFilter &) const
- virtual void printLegend (QPainter *, const QRect &) const

Static Protected Member Functions

- static bool axisValid (int axisId)

12.49.1 Detailed Description

A 2-D plotting widget. QwtPlot is a widget for plotting two-dimensional graphs. An unlimited number of plot items can be displayed on its canvas. Plot items might be curves (QwtPlotCurve), markers (QwtPlotMarker), the grid (QwtPlotGrid), or anything else derived from QwtPlotItem. A plot can have up to four axes, with each plot item attached to an x- and a y axis. The scales at the axes can be explicitely set (QwtScaleDiv), or are calculated from the plot items, using algorithms (QwtScaleEngine) which can be configured separately for each axis.

Example

The following example shows (schematically) the most simple way to use QwtPlot. By default, only the left and bottom axes are visible and their scales are computed automatically.

```
#include <qwt_plot.h>
#include <qwt_plot_curve.h>

QwtPlot *myPlot = new QwtPlot("Two Curves", parent);

// add curves
QwtPlotCurve *curve1 = new QwtPlotCurve("Curve 1");
QwtPlotCurve *curve2 = new QwtPlotCurve("Curve 2");

// copy the data into the curves
curve1->setData(...);
curve2->setData(...);

curve1->attach(myPlot);
curve2->attach(myPlot);

// finally, refresh the plot
myPlot->replot();
```

12.49.2 Member Enumeration Documentation

12.49.2.1 enum QwtPlot::Axis

Axis index

- yLeft
- yRight
- xBottom
- xTop

12.49.2.2 enum QwtPlot::LegendPosition

Position of the legend, relative to the canvas.

- LeftLegend

  The legend will be left from the yLeft axis.
- RightLegend

  The legend will be right from the yLeft axis.
- BottomLegend

  The legend will be right below the xBottom axis.
- TopLegend

  The legend will be between xTop axis and the title.
- ExternalLegend

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

Note

In case of ExternalLegend, the legend is not printed by print().

See also

insertLegend()

12.49.3 Constructor & Destructor Documentation

12.49.3.1 QwtPlot::QwtPlot (QWidget * *parent* = NULL) [explicit]

Constructor.

Parameters

parent Parent widget

12.49.3.2 QwtPlot::QwtPlot (const QwtText & *title*, QWidget * *parent* = NULL) [explicit]

Constructor.

Parameters

title Title text

parent Parent widget

12.49.3.3 QwtPlot::~QwtPlot () [virtual]

Destructor.

12.49.4 Member Function Documentation

12.49.4.1 void QwtPlot::applyProperties (const QString &)

This method is intended for manipulating the plot widget from a specific editor in the Qwt designer plugin.

Warning

The plot editor has never been implemented.

12.49.4.2 void QwtPlot::autoRefresh () [slot]

Replots the plot if QwtPlot::autoReplot() is `true`.

12.49.4.3 bool QwtPlot::autoReplot () const

Returns

true if the autoReplot option is set.

12.49.4.4 bool QwtPlot::axisAutoScale (int *axisId*) const

Returns

`true` if autoscaling is enabled

Parameters

axisId axis index

12.49.4.5 bool QwtPlot::axisEnabled (int *axisId*) const

Returns

`true` if a specified axis is enabled

Parameters

axisId axis index

12.49.4.6 QFont QwtPlot::axisFont (int *axisId*) const

Returns

the font of the scale labels for a specified axis

Parameters

axisId axis index

12.49.4.7 int QwtPlot::axisMaxMajor (int *axisId*) const

Returns

the maximum number of major ticks for a specified axis

Parameters

axisId axis index sa setAxisMaxMajor()

12.49.4.8 int QwtPlot::axisMaxMinor (int *axisId*) const

Returns

the maximum number of minor ticks for a specified axis

Parameters

axisId axis index sa setAxisMaxMinor()

12.49.4.9 QwtScaleDiv * QwtPlot::axisScaleDiv (int *axisId*)

Return the scale division of a specified axis.

axisScaleDiv(axisId)->lowerBound(), axisScaleDiv(axisId)->upperBound() are the current limits of the axis scale.

Parameters

axisId axis index

Returns

Scale division

See also

QwtScaleDiv, setAxisScaleDiv()

12.49.4.10 const QwtScaleDiv * QwtPlot::axisScaleDiv (int *axisId*) const

Return the scale division of a specified axis.

axisScaleDiv(axisId)->lowerBound(), axisScaleDiv(axisId)->upperBound() are the current limits of the axis scale.

Parameters

axisId axis index

Returns

Scale division

See also

QwtScaleDiv, setAxisScaleDiv()

12.49.4.11 QwtScaleDraw * QwtPlot::axisScaleDraw (int *axisId*)

Returns

the scale draw of a specified axis

Parameters

axisId axis index

Returns

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

See also

QwtScaleDraw

12.49.4.12 const QwtScaleDraw * QwtPlot::axisScaleDraw (int *axisId*) const

Returns

the scale draw of a specified axis

Parameters

axisId axis index

Returns

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

See also

QwtScaleDraw

12.49.4.13 const QwtScaleEngine * QwtPlot::axisScaleEngine (int *axisId*) const

Parameters

axisId axis index

Returns

Scale engine for a specific axis

12.49.4.14 QwtScaleEngine * QwtPlot::axisScaleEngine (int *axisId*)

Parameters

axisId axis index

Returns

Scale engine for a specific axis

12.49.4.15 double QwtPlot::axisStepSize (int *axisId*) const

Return the step size parameter, that has been set in setAxisScale. This doesn't need to be the step size of the current scale.

Parameters

axisId axis index

Returns

step size parameter value

See also

setAxisScale()

12.49.4.16 QwtText QwtPlot::axisTitle (int *axisId*) const

Returns

the title of a specified axis

Parameters

axisId axis index

12.49.4.17 bool QwtPlot::axisValid (int *axisId*) [static, protected]

Returns

`true` if the specified axis exists, otherwise `false`

Parameters

axisId axis index

12.49.4.18 QwtScaleWidget * QwtPlot::axisWidget (int *axisId*)

Returns

specified axis, or NULL if axisId is invalid.

Parameters

axisId axis index

12.49.4.19 const QwtScaleWidget * QwtPlot::axisWidget (int *axisId*) const

Returns

specified axis, or NULL if axisId is invalid.

Parameters

axisId axis index

12.49.4.20 const QwtPlotCanvas * QwtPlot::canvas () const

Returns

the plot's canvas

12.49.4.21 QwtPlotCanvas * QwtPlot::canvas ()

Returns

the plot's canvas

12.49.4.22 const QColor & QwtPlot::canvasBackground () const

Nothing else than: canvas()->palette().color(QPalette::Normal, QColorGroup::Background);

Returns

the background color of the plotting area.

12.49.4.23 int QwtPlot::canvasLineWidth () const

Nothing else than: canvas()->lineWidth(), left for compatibility only.

Returns

the border width of the plotting area

12.49.4.24 QwtScaleMap QwtPlot::canvasMap (int *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()

12.49.4.25 void QwtPlot::clear () [virtual, slot]

Remove all curves and markers

Deprecated

Use QwtPlotDeict::detachItems instead

12.49.4.26 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()

12.49.4.27 void QwtPlot::drawItems (QPainter * *painter*, const QRect & *rect*, const QwtScaleMap *map*[axisCnt], const QwtPlotPrintFilter & *pfilter*) const [protected, virtual]

Redraw the canvas items.

Parameters

painter Painter used for drawing

rect Bounding rectangle where to paint

map QwtPlot::axisCnt maps, mapping between plot and paint device coordinates

pfilter Plot print filter

12.49.4.28 void QwtPlot::enableAxis (int *axisId*, bool *tf* = true)

Enable or disable a specified axis.

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

Only xBottom and yLeft are enabled by default.

Parameters

axisId axis index

tf `true` (enabled) or `false` (disabled)

12.49.4.29 bool QwtPlot::event (QEvent * *e*) [virtual]

Adds handling of layout requests.

12.49.4.30 QString QwtPlot::grabProperties () const

This method is intended for manipulating the plot widget from a specific editor in the Qwt designer plugin.

Warning

The plot editor has never been implemented.

12.49.4.31 void QwtPlot::insertLegend (QwtLegend * *legend*, QwtPlot::LegendPosition *pos* = QwtPlot::RightLegend, double *ratio* = -1.0)

Insert a legend.

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

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

Parameters

legend Legend

pos The legend's position. For top/left position the number of colums will be limited to 1, otherwise it will be set to unlimited.

ratio Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrinked if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also

legend(), QwtPlotLayout::legendPosition(), QwtPlotLayout::setLegendPosition()

12.49.4.32 double QwtPlot::invTransform (int *axisId*, int *pos*) const

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

Parameters

axisId axis index

pos position

Warning

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

12.49.4.33 const QwtLegend * QwtPlot::legend () const

Returns

the plot's legend

See also

insertLegend()

12.49.4.34 QwtLegend * QwtPlot::legend ()

Returns

the plot's legend

See also

insertLegend()

12.49.4.35 void QwtPlot::legendChecked (QwtPlotItem * *plotItem*, bool *on*) [signal]

A signal which is emitted when the user has clicked on a legend item, which is in QwtLegend::CheckableItem mode

Parameters

plotItem Corresponding plot item of the selected legend item

on True when the legen item is checked

Note

clicks are disabled as default

See also

QwtLegend::setItemMode(), QwtLegend::itemMode()

12.49.4.36 void QwtPlot::legendClicked (QwtPlotItem * *plotItem*) [signal]

A signal which is emitted when the user has clicked on a legend item, which is in QwtLegend::ClickableItem mode.

Parameters

plotItem Corresponding plot item of the selected legend item

Note

clicks are disabled as default

See also

QwtLegend::setItemMode(), QwtLegend::itemMode()

12.49.4.37 void QwtPlot::legendItemChecked (bool *on*) [protected, virtual, slot]

Called internally when the legend has been checked Emits a legendClicked() signal.

12.49.4.38 void QwtPlot::legendItemClicked () [protected, virtual, slot]

Called internally when the legend has been clicked on. Emits a legendClicked() signal.

12.49.4.39 int QwtPlot::margin () const

Returns

margin

See also

setMargin(), QwtPlotLayout::margin(), plotLayout()

12.49.4.40 QSize QwtPlot::minimumSizeHint () const [virtual]

Return a minimum size hint.

12.49.4.41 const QwtPlotLayout * QwtPlot::plotLayout () const

Returns

the plot's titel label.

12.49.4.42 QwtPlotLayout * QwtPlot::plotLayout ()

Returns

the plot's title

12.49.4.43 void QwtPlot::polish () [virtual]

Polish.

12.49.4.44 void QwtPlot::print (QPainter * *painter*, const QRect & *plotRect*, const QwtPlotPrintFilter & *pfilter* = QwtPlotPrintFilter ()) const [virtual]

Paint the plot into a given rectangle. Paint the contents of a QwtPlot instance into a given rectangle.

Parameters

painter Painter

plotRect Bounding rectangle

pfilter Print filter

See also

QwtPlotPrintFilter

12.49.4.45 void QwtPlot::print (QPaintDevice & *paintDev*, const QwtPlotPrintFilter & *pfilter* = QwtPlotPrintFilter ()) const

Print the plot to a `QPaintDevice` (`QPrinter`) This function prints the contents of a QwtPlot instance to `QPaintDevice` object. The size is derived from its device metrics.

Parameters

paintDev device to paint on, often a printer

pfilter print filter

See also

QwtPlotPrintFilter

12.49.4.46 void QwtPlot::printCanvas (QPainter * *painter*, const QRect & *boundingRect*, const QRect & *canvasRect*, const QwtScaleMap *map*[axisCnt], const QwtPlotPrintFilter & *pfilter*) const [protected, virtual]

Print the canvas into a given rectangle.

Parameters

painter Painter

map Maps mapping between plot and paint device coordinates

boundingRect Bounding rectangle

canvasRect Canvas rectangle

pfilter Print filter

See also

QwtPlotPrintFilter

12.49.4.47 void QwtPlot::printLegend (QPainter * *painter*, const QRect & *rect*) const [protected, virtual]

Print the legend into a given rectangle.

Parameters

painter Painter

rect Bounding rectangle

12.49.4.48 void QwtPlot::printLegendItem (QPainter * *painter*, const QWidget * *w*, const QRect & *rect*) const [protected, virtual]

Print the legend item into a given rectangle.

Parameters

painter Painter

w Widget representing a legend item

rect Bounding rectangle

12.49.4.49 void QwtPlot::printScale (QPainter * *painter*, int *axisId*, int *startDist*, int *endDist*, int *baseDist*, const QRect & *rect*) const [protected, virtual]

Paint a scale into a given rectangle. Paint the scale into a given rectangle.

Parameters

painter Painter

axisId Axis

startDist Start border distance

endDist End border distance

baseDist Base distance

rect Bounding rectangle

12.49.4.50 void QwtPlot::printTitle (QPainter * *painter*, const QRect & *rect*) const [protected, virtual]

Print the title into a given rectangle.

Parameters

painter Painter

rect Bounding rectangle

12.49.4.51 void QwtPlot::replot () [virtual, slot]

Redraw the plot.

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

See also

setAutoReplot()

Warning

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

12.49.4.52 void QwtPlot::resizeEvent (QResizeEvent * *e*) [protected, virtual]

Resize and update internal layout

Parameters

e Resize event

12.49.4.53 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()

12.49.4.54 void QwtPlot::setAxisAutoScale (int *axisId*)

Enable autoscaling for a specified axis.

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

Parameters

axisId axis index

See also

setAxisScale(), setAxisScaleDiv()

12.49.4.55 void QwtPlot::setAxisFont (int *axisId*, const QFont & *f*)

Change the font of an axis.

Parameters

axisId axis index

f font

Warning

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

12.49.4.56 void QwtPlot::setAxisLabelAlignment (int *axisId*, Qt::Alignment *alignment*)

Change the alignment of the tick labels

Parameters

axisId axis index

alignment Or'd Qt::AlignmentFlags <see qnamespace.h>

See also

QwtScaleDraw::setLabelAlignment()

12.49.4.57 void QwtPlot::setAxisLabelRotation (int *axisId*, double *rotation*)

Rotate all tick labels

Parameters

axisId axis index

rotation Angle in degrees. When changing the label rotation, the label alignment might be adjusted too.

See also

QwtScaleDraw::setLabelRotation(), setAxisLabelAlignment()

12.49.4.58 void QwtPlot::setAxisMaxMajor (int *axisId*, int *maxMajor*)

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

Parameters

axisId axis index

maxMajor maximum number of major steps

See also

axisMaxMajor()

12.49.4.59 void QwtPlot::setAxisMaxMinor (int *axisId*, int *maxMinor*)

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

Parameters

axisId axis index

maxMinor maximum number of minor steps

See also

axisMaxMinor()

12.49.4.60 void QwtPlot::setAxisScale (int *axisId*, double *min*, double *max*, double *stepSize* = 0)

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

Parameters

axisId axis 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

setAxisMaxMajor(), setAxisAutoScale()

12.49.4.61 void QwtPlot::setAxisScaleDiv (int *axisId*, const QwtScaleDiv & *scaleDiv*)

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

Parameters

axisId axis index

scaleDiv Scale division

See also

setAxisScale(), setAxisAutoScale()

12.49.4.62 void QwtPlot::setAxisScaleDraw (int *axisId*, QwtScaleDraw * *scaleDraw*)

Set a scale draw.

Parameters

axisId axis index

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.

12.49.4.63 void QwtPlot::setAxisScaleEngine (int *axisId*, QwtScaleEngine * *scaleEngine*)

Change the scale engine for an axis

Parameters

axisId axis index

scaleEngine Scale engine

See also

axisScaleEngine()

12.49.4.64 void QwtPlot::setAxisTitle (int *axisId*, const QwtText & *title*)

Change the title of a specified axis.

Parameters

axisId axis index

title axis title

12.49.4.65 void QwtPlot::setAxisTitle (int *axisId*, const QString & *title*)

Change the title of a specified axis.

Parameters

axisId axis index

title axis title

12.49.4.66 void QwtPlot::setCanvasBackground (const QColor & *c*)

Change the background of the plotting area.

Sets c to QColorGroup::Background of all colorgroups of the palette of the canvas. Using canvas()->setPalette() is a more powerful way to set these colors.

Parameters

c new background color

12.49.4.67 void QwtPlot::setCanvasLineWidth (int *w*)

Change the border width of the plotting area Nothing else than canvas()->setLineWidth(w), left for compatibility only.

Parameters

w new border width

12.49.4.68 void QwtPlot::setMargin (int *margin*)

Change the margin of the plot. The margin is the space around all components.

Parameters

margin new margin

See also

QwtPlotLayout::setMargin(), margin(), plotLayout()

12.49.4.69 void QwtPlot::setTitle (const QwtText & *title*)

Change the plot's title

Parameters

title New title

12.49.4.70 void QwtPlot::setTitle (const QString & *title*)

Change the plot's title

Parameters

title New title

12.49.4.71 QSize QwtPlot::sizeHint () const [virtual]

Return sizeHint

See also

minimumSizeHint()

12.49.4.72 QwtText QwtPlot::title () const

Returns

the plot's title

12.49.4.73 const QwtTextLabel * QwtPlot::titleLabel () const

Returns

the plot's titel label.

12.49.4.74 QwtTextLabel * QwtPlot::titleLabel ()

Returns

the plot's titel label.

12.49.4.75 int QwtPlot::transform (int *axisId*, double *value*) const

Transform a value into a coordinate in the plotting region.

Parameters

axisId axis index

value value

Returns

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

12.49.4.76 void QwtPlot::updateAxes ()

Rebuild the scales.

12.49.4.77 void QwtPlot::updateLayout () `[virtual]`

Adjust plot content to its current size.

See also

resizeEvent()

12.49.4.78 void QwtPlot::updateTabOrder () `[protected, virtual]`

Update the focus tab order

The order is changed so that the canvas will be in front of the first legend item, or behind the last legend item - depending on the position of the legend.

12.50 QwtPlotCanvas Class Reference

Canvas of a QwtPlot.

`#include <qwt_plot_canvas.h>`

Public Types

- enum PaintAttribute {
  PaintCached = 1,
  PaintPacked = 2 }
- enum FocusIndicator {
  NoFocusIndicator,
  CanvasFocusIndicator,
  ItemFocusIndicator }

Public Member Functions

- QwtPlotCanvas (QwtPlot ∗)
- virtual ∼QwtPlotCanvas ()
- QwtPlot ∗ plot ()
- const QwtPlot ∗ plot () const
- void setFocusIndicator (FocusIndicator)
- FocusIndicator focusIndicator () const
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- QPixmap ∗ paintCache ()
- const QPixmap ∗ paintCache () const
- void invalidatePaintCache ()
- void replot ()

Protected Member Functions

- virtual void hideEvent (QHideEvent ∗)
- virtual void paintEvent (QPaintEvent ∗)
- virtual void drawContents (QPainter ∗)
- virtual void drawFocusIndicator (QPainter ∗)
- void drawCanvas (QPainter ∗painter=NULL)

12.50.1 Detailed Description

Canvas of a QwtPlot.

See also

QwtPlot

12.50.2 Member Enumeration Documentation

12.50.2.1 enum QwtPlotCanvas::FocusIndicator

Focus indicator.

- NoFocusIndicator

  Don't paint a focus indicator

- CanvasFocusIndicator

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

- ItemFocusIndicator

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

See also

setFocusIndicator(), focusIndicator(), paintFocus()

12.50.2.2 enum QwtPlotCanvas::PaintAttribute

Paint attributes.

- PaintCached

  Paint double buffered and reuse the content of the pixmap buffer for some spontaneous repaints that happen when a plot gets unhidden, deiconified or changes the focus. Disabling the cache will improve the performance for incremental paints (using QwtPlotCurve::draw).

- PaintPacked

  Suppress system background repaints and paint it together with the canvas contents. Painting packed might avoid flickering for expensive repaints, when there is a notable gap between painting the background and the plot contents.

The default setting enables PaintCached and PaintPacked

See also

setPaintAttribute(), testPaintAttribute(), paintCache()

12.50.3 Constructor & Destructor Documentation

12.50.3.1 QwtPlotCanvas::QwtPlotCanvas (QwtPlot * *plot*) `[explicit]`

Sets a cross cursor, enables QwtPlotCanvas::PaintCached.

12.50.3.2 QwtPlotCanvas::~QwtPlotCanvas () [virtual]

Destructor.

12.50.4 Member Function Documentation

12.50.4.1 void QwtPlotCanvas::drawCanvas (QPainter * *painter* = NULL) [protected]

Draw the the canvas

Paints all plot items to the contentsRect(), using QwtPlot::drawCanvas and updates the paint cache.

Parameters

painter Painter

See also

QwtPlot::drawCanvas(), setPaintAttributes(), testPaintAttributes()

12.50.4.2 void QwtPlotCanvas::drawContents (QPainter * *painter*) [protected, virtual]

Redraw the canvas, and focus rect

Parameters

painter Painter

12.50.4.3 void QwtPlotCanvas::drawFocusIndicator (QPainter * *painter*) [protected, virtual]

Draw the focus indication

Parameters

painter Painter

12.50.4.4 QwtPlotCanvas::FocusIndicator QwtPlotCanvas::focusIndicator () const

Returns

Focus indicator

See also

FocusIndicator, setFocusIndicator()

12.50.4.5 void QwtPlotCanvas::hideEvent (QHideEvent * *event*) [protected, virtual]

Hide event

Parameters

event Hide event

12.50.4.6 void QwtPlotCanvas::invalidatePaintCache ()

Invalidate the internal paint cache.

12.50.4.7 const QPixmap * QwtPlotCanvas::paintCache () const

Return the paint cache, might be null.

12.50.4.8 QPixmap * QwtPlotCanvas::paintCache ()

Return the paint cache, might be null.

12.50.4.9 void QwtPlotCanvas::paintEvent (QPaintEvent * *event*) [protected, virtual]

Paint event

Parameters

event Paint event

12.50.4.10 const QwtPlot * QwtPlotCanvas::plot () const

Return parent plot widget.

12.50.4.11 QwtPlot * QwtPlotCanvas::plot ()

Return parent plot widget.

12.50.4.12 void QwtPlotCanvas::replot ()

Invalidate the paint cache and repaint the canvas

See also

invalidatePaintCache()

12.50.4.13 void QwtPlotCanvas::setFocusIndicator (FocusIndicator *focusIndicator*)

Set the focus indicator

See also

FocusIndicator, focusIndicator()

12.50.4.14 void QwtPlotCanvas::setPaintAttribute (PaintAttribute *attribute*, bool *on* = `true`)

Changing the paint attributes.

Parameters

attribute Paint attribute

on On/Off

The default setting enables PaintCached and PaintPacked

See also

testPaintAttribute(), drawCanvas(), drawContents(), paintCache()

12.50.4.15 bool QwtPlotCanvas::testPaintAttribute (PaintAttribute *attribute*) const

Test wether a paint attribute is enabled

Parameters

attribute Paint attribute

Returns

true if the attribute is enabled

See also

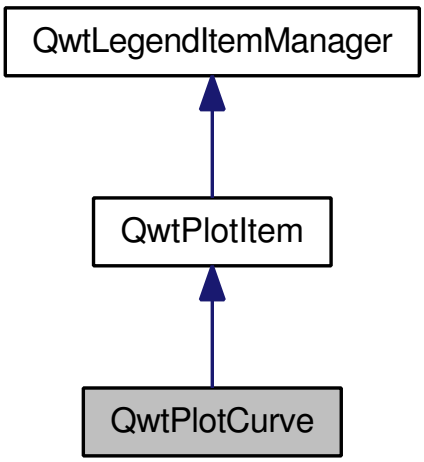
setPaintAttribute()

12.51 QwtPlotCurve Class Reference

A plot item, that represents a series of points.

`#include <qwt_plot_curve.h>`

Inheritance diagram for QwtPlotCurve:



Public Types

- enum CurveType {
  Yfx,
  Xfy }
- enum CurveStyle {
  NoCurve,
  Lines,
  Sticks,
  Steps,
  Dots,
  UserCurve = 100 }
- enum CurveAttribute {
  Inverted = 1,
  Fitted = 2 }
- enum PaintAttribute {
  PaintFiltered = 1,
  ClipPolygons = 2 }

Public Member Functions

- QwtPlotCurve ()
- QwtPlotCurve (const QwtText &title)
- QwtPlotCurve (const QString &title)
- virtual ~QwtPlotCurve ()
- virtual int rtti () const
- void setCurveType (CurveType)
- CurveType curveType () const
- void setPaintAttribute (PaintAttribute, bool on=true)

- bool testPaintAttribute (PaintAttribute) const
- void setRawData (const double *x, const double *y, int size)
- void setData (const double *xData, const double *yData, int size)
- void setData (const QwtArray< double > &xData, const QwtArray< double > &yData)
- void setData (const QPolygonF &data)
- void setData (const QwtData &data)
- int closestPoint (const QPoint &pos, double *dist=NULL) const
- QwtData & data ()
- const QwtData & data () const
- int dataSize () const
- double x (int i) const
- double y (int i) const
- virtual QwtDoubleRect boundingRect () const
- double minXValue () const
- double maxXValue () const
- double minYValue () const
- double maxYValue () const
- void setCurveAttribute (CurveAttribute, bool on=true)
- bool testCurveAttribute (CurveAttribute) const
- void setPen (const QPen &)
- const QPen & pen () const
- void setBrush (const QBrush &)
- const QBrush & brush () const
- void setBaseline (double ref)
- double baseline () const
- void setStyle (CurveStyle style)
- CurveStyle style () const
- void setSymbol (const QwtSymbol &s)
- const QwtSymbol & symbol () const
- void setCurveFitter (QwtCurveFitter *)
- QwtCurveFitter * curveFitter () const
- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &) const
- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void draw (int from, int to) const
- virtual void updateLegend (QwtLegend *) const

Protected Member Functions

- void init ()
- virtual void drawCurve (QPainter *p, int style, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- virtual void drawSymbols (QPainter *p, const QwtSymbol &, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void drawLines (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void drawSticks (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void drawDots (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const

- void drawSteps (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, int from, int to) const
- void fillCurve (QPainter *, const QwtScaleMap &, const QwtScaleMap &, QwtPolygon &) const
- void closePolyline (const QwtScaleMap &, const QwtScaleMap &, QwtPolygon &) const

12.51.1 Detailed Description

A plot item, that represents a series of points. A curve is the representation of a series of points in the x-y plane. It supports different display styles, interpolation (f.e. spline) and symbols.

Usage

a) Assign curve properties When a curve is created, it is configured to draw black solid lines with in Lines style and no symbols. You can change this by calling setPen(), setStyle() and setSymbol().

b) Connect/Assign data. QwtPlotCurve gets its points using a QwtData object offering a bridge to the real storage of the points (like QAbstractItemModel). There are several convenience classes derived from QwtData, that also store the points inside (like QStandardItemModel). QwtPlotCurve also offers a couple of variations of setData(), that build QwtData objects from arrays internally.

c) Attach the curve to a plot See QwtPlotItem::attach()

Example:

see examples/bode

See also

QwtPlot, QwtData, QwtSymbol, QwtScaleMap

12.51.2 Member Enumeration Documentation

12.51.2.1 enum QwtPlotCurve::CurveAttribute

Attribute for drawing the curve

- Fitted (in combination with the Lines QwtPlotCurve::CurveStyle only)

  A QwtCurveFitter tries to interpolate/smooth the curve, before it is painted. Note that curve fitting requires temorary memory for calculating coefficients and additional points. If painting in Fitted mode is slow it might be better to fit the points, before they are passed to QwtPlotCurve.

- Inverted

  For Steps only. Draws a step function from the right to the left.

See also

setCurveAttribute(), testCurveAttribute(), curveFitter()

12.51.2.2 enum QwtPlotCurve::CurveStyle

Curve styles.

- NoCurve

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

- Lines

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

- Sticks

  Draw vertical(Yfx) or horizontal(Xfy) sticks from a baseline which is defined by setBaseline().

- Steps

  Connect the points with a step function. The step function is drawn from the left to the right or vice versa, depending on the 'Inverted' attribute.

- Dots

  Draw dots at the locations of the data points. Note: This is different from a dotted line (see setPen()), and faster as a curve in NoStyle style and a symbol painting a point.

- UserCurve

  Styles >= UserCurve are reserved for derived classes of QwtPlotCurve that overload drawCurve() with additional application specific curve types.

See also

setStyle(), style()

12.51.2.3 enum QwtPlotCurve::CurveType

Curve type.

- Yfx

  Draws y as a function of x (the default). The baseline is interpreted as a horizontal line with y = baseline().

- Xfy

  Draws x as a function of y. The baseline is interpreted as a vertical line with x = baseline().

The baseline is used for aligning the sticks, or filling the curve with a brush.

See also

setCurveType(), curveType(), baseline() brush()

12.51.2.4 enum QwtPlotCurve::PaintAttribute

Attributes to modify the drawing algorithm.

- PaintFiltered

  Tries to reduce the data that has to be painted, by sorting out duplicates, or paintings outside the visible area. Might have a notable impact on curves with many close points. Only a couple of very basic filtering algos are implemented.

- ClipPolygons

  Clip polygons before painting them. In situations, where points are far outside the visible area (f.e when zooming deep) this might be a substantial improvement for the painting performance (especially on Windows).

The default is, that no paint attributes are enabled.

See also

setPaintAttribute(), testPaintAttribute()

12.51.3 Constructor & Destructor Documentation

12.51.3.1 QwtPlotCurve::QwtPlotCurve () [explicit]

Constructor.

12.51.3.2 QwtPlotCurve::QwtPlotCurve (const QwtText & *title*) [explicit]

Constructor

Parameters

title Title of the curve

12.51.3.3 QwtPlotCurve::QwtPlotCurve (const QString & *title*) [explicit]

Constructor

Parameters

title Title of the curve

12.51.3.4 QwtPlotCurve::~QwtPlotCurve () [virtual]

Destructor.

12.51.4 Member Function Documentation

12.51.4.1 double QwtPlotCurve::baseline () const

Return the value of the baseline

See also

setBaseline()

12.51.4.2 QwtDoubleRect QwtPlotCurve::boundingRect () const [virtual]

Returns the bounding rectangle of the curve data. If there is no bounding rect, like for empty data the rectangle is invalid.

See also

QwtData::boundingRect(), QwtDoubleRect::isValid()

Reimplemented from QwtPlotItem.

12.51.4.3 const QBrush & QwtPlotCurve::brush () const

Return the brush used to fill the area between lines and the baseline.

See also

setBrush(), setBaseline(), baseline()

12.51.4.4 void QwtPlotCurve::closePolyline (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, QwtPolygon & *pa*) const [protected]

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

Parameters

xMap X map

yMap Y map

pa Polygon to be completed

12.51.4.5 int QwtPlotCurve::closestPoint (const QPoint & *pos*, double * *dist* = NULL) const

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 clostest curve point

Returns

Index of the closest curve point, or -1 if none can be found (f.e when the curve has no points)

Note

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

12.51.4.6 QwtCurveFitter * QwtPlotCurve::curveFitter () const

Get the curve fitter. If curve fitting is disabled NULL is returned.

Returns

Curve fitter

12.51.4.7 QwtPlotCurve::CurveType QwtPlotCurve::curveType () const

Return the curve type

See also

CurveType, setCurveType()

12.51.4.8 const QwtData & QwtPlotCurve::data () const [inline]

Returns

the the curve data

12.51.4.9 QwtData & QwtPlotCurve::data () [inline]

Returns

the the curve data

12.51.4.10 int QwtPlotCurve::dataSize () const

Return the size of the data arrays

See also

setData()

12.51.4.11 void QwtPlotCurve::draw (int *from*, int *to*) const

Draw a set of points of a curve.

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

Setting plot()->canvas()->setAttribute(Qt::WA_PaintOutsidePaintEvent, true); will result in faster painting, if the paint engine of the canvas widget supports this feature.

Parameters

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()

12.51.4.12 void QwtPlotCurve::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [virtual]

Draw an interval of the curve.

Parameters

painter Painter

xMap maps x-values into pixel coordinates.

yMap maps y-values into pixel 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(),

12.51.4.13 void QwtPlotCurve::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the complete curve.

Parameters

painter Painter

xMap Maps x-values into pixel coordinates.

yMap Maps y-values into pixel coordinates.

See also

drawCurve(), drawSymbols()

Implements QwtPlotItem.

12.51.4.14 void QwtPlotCurve::drawCurve (QPainter * *painter*, int *style*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [protected, virtual]

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

Parameters

painter Painter

style curve style, see QwtPlotCurve::CurveStyle

xMap x map

yMap y map

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()

12.51.4.15 void QwtPlotCurve::drawDots (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [protected]

Draw dots

Parameters

painter Painter

xMap x map

yMap y map

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()

12.51.4.16 void QwtPlotCurve::drawLines (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [protected]

Draw lines.

If the CurveAttribute Fitted is enabled a QwtCurveFitter tries to interpolate/smooth the curve, before it is painted.

Parameters

painter Painter

xMap x map

yMap y map

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()

12.51.4.17 void QwtPlotCurve::drawSteps (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [protected]

Draw step function

The direction of the steps depends on Inverted attribute.

Parameters

painter Painter

xMap x map

yMap y map

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()

12.51.4.18 void QwtPlotCurve::drawSticks (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [protected]

Draw sticks

Parameters

painter Painter

xMap x map

yMap y map

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()

12.51.4.19 void QwtPlotCurve::drawSymbols (QPainter * *painter*, const QwtSymbol & *symbol*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, int *from*, int *to*) const [protected, virtual]

Draw symbols.

Parameters

painter Painter

symbol Curve symbol

xMap x map

yMap y map

from index of the first point to be painted

to index of the last point to be painted

See also

setSymbol(), draw(), drawCurve()

12.51.4.20 void QwtPlotCurve::fillCurve (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, QwtPolygon & *pa*) const [protected]

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

Parameters

painter Painter

xMap x map

yMap y map

pa Polygon

See also

setBrush(), setBaseline(), setCurveType()

12.51.4.21 void QwtPlotCurve::init () [protected]

Initialize data members.

12.51.4.22 double QwtPlotCurve::maxXValue () const [inline]

boundingRect().right()

12.51.4.23 double QwtPlotCurve::maxYValue () const [inline]

boundingRect().bottom()

12.51.4.24 double QwtPlotCurve::minXValue () const [inline]

boundingRect().left()

12.51.4.25 double QwtPlotCurve::minYValue () const [inline]

boundingRect().top()

12.51.4.26 const QPen & QwtPlotCurve::pen () const

Return the pen used to draw the lines.

See also

setPen(), brush()

12.51.4.27 int QwtPlotCurve::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotCurve

Reimplemented from QwtPlotItem.

12.51.4.28 void QwtPlotCurve::setBaseline (double *reference*)

Set the value of the baseline.

The baseline is needed for filling the curve with a brush or the Sticks drawing style. The default value is 0.0. The interpretation of the baseline depends on the CurveType. With QwtPlotCurve::Yfx, the baseline is interpreted as a horizontal line at y = baseline(), with QwtPlotCurve::Yfy, it is interpreted as a vertical line at x = baseline().

Parameters

reference baseline

See also

baseline(), setBrush(), setStyle(), setCurveType()

12.51.4.29 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()

12.51.4.30 void QwtPlotCurve::setCurveAttribute (CurveAttribute *attribute*, bool *on* = `true`)

Specify an attribute for drawing the curve

Parameters

attribute Curve attribute

on On/Off

/sa CurveAttribute, testCurveAttribute(), setCurveFitter()

12.51.4.31 void QwtPlotCurve::setCurveFitter (QwtCurveFitter * *curveFitter*)

Assign a curve fitter setCurveFitter(NULL) disables curve fitting.

Parameters

curveFitter Curve fitter

12.51.4.32 void QwtPlotCurve::setCurveType (CurveType *curveType*)

Assign the curve type

Parameters

curveType Yfx or Xfy

See also

CurveType, curveType()

12.51.4.33 void QwtPlotCurve::setData (const QwtData & *data*)

Initialize data with a pointer to QwtData.

Parameters

data Data

See also

QwtData::copy()

12.51.4.34 void QwtPlotCurve::setData (const QPolygonF & *data*)

Initialize data with an array of points (explicitly shared).

Parameters

data Data

Note

Internally the data is stored in a QwtPolygonFData object

12.51.4.35 void QwtPlotCurve::setData (const QwtArray< double > & *xData*, const QwtArray< double > & *yData*)

Initialize data with x- and y-arrays (explicitly shared) (Builds an QwtArrayData object internally)

Parameters

xData x data

yData y data

Note

Internally the data is stored in a QwtArrayData object

12.51.4.36 void QwtPlotCurve::setData (const double * *xData*, const double * *yData*, int *size*)

Set data by copying x- and y-values from specified memory blocks. Contrary to setCurveRawData(), this function makes a 'deep copy' of the data.

Parameters

xData Pointer to x values

yData Pointer to y values

size Size of xData and yData

Note

Internally the data is stored in a QwtArrayData object

12.51.4.37 void QwtPlotCurve::setPaintAttribute (PaintAttribute *attribute*, bool *on* = `true`)

Specify an attribute how to draw the curve

Parameters

attribute Paint attribute

on On/Off /sa PaintAttribute, testPaintAttribute()

12.51.4.38 void QwtPlotCurve::setPen (const QPen & *pen*)

Assign a pen

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

pen New pen

See also

pen(), brush(), QwtPainter::scaledPen()

12.51.4.39 void QwtPlotCurve::setRawData (const double * *xData*, const double * *yData*, int *size*)

Initialize the data by pointing to memory blocks which are not managed by QwtPlotCurve.

setRawData is provided for efficiency. It is important to keep the pointers during the lifetime of the underlying QwtCPointerData class.

Parameters

xData pointer to x data

yData pointer to y data

size size of x and y

Note

Internally the data is stored in a QwtCPointerData object

12.51.4.40 void QwtPlotCurve::setStyle (CurveStyle *style*)

Set the curve's drawing style

Parameters

style Curve style

See also

CurveStyle, style()

12.51.4.41 void QwtPlotCurve::setSymbol (const QwtSymbol & *symbol*)

Assign a symbol.

Parameters

symbol Symbol

See also

symbol()

12.51.4.42 QwtPlotCurve::CurveStyle QwtPlotCurve::style () const

Return the current style

See also

CurveStyle, setStyle()

12.51.4.43 const QwtSymbol & QwtPlotCurve::symbol () const

Return the current symbol.

See also

setSymbol()

12.51.4.44 bool QwtPlotCurve::testCurveAttribute (CurveAttribute *attribute*) const

Returns

true, if attribute is enabled

See also

CurveAttribute, setCurveAttribute()

12.51.4.45 bool QwtPlotCurve::testPaintAttribute (PaintAttribute *attribute*) const

Return the current paint attributes.

See also

PaintAttribute, setPaintAttribute()

12.51.4.46 void QwtPlotCurve::updateLegend (QwtLegend * *legend*) const [virtual]

Update the widget that represents the curve on the legend.

Reimplemented from QwtPlotItem.

12.51.4.47 double QwtPlotCurve::x (int *i*) const [inline]

Parameters

i index

Returns

x-value at position i

12.51.4.48 double QwtPlotCurve::y (int *i*) const [inline]

Parameters

i index

Returns

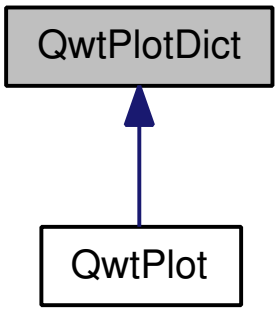
y-value at position i

12.52 QwtPlotDict Class Reference

A dictionary for plot items.

`#include <qwt_plot_dict.h>`

Inheritance diagram for QwtPlotDict:



Public Member Functions

- QwtPlotDict ()
- ~QwtPlotDict ()
- void setAutoDelete (bool)
- bool autoDelete () const
- const QwtPlotItemList & itemList () const
- void detachItems (int rtti=QwtPlotItem::Rtti_PlotItem, bool autoDelete=true)

Friends

- class QwtPlotItem

12.52.1 Detailed Description

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

See also

QwtPlotItem::attach(), QwtPlotItem::detach(), QwtPlotItem::z()

12.52.2 Constructor & Destructor Documentation

12.52.2.1 QwtPlotDict::QwtPlotDict () [explicit]

Constructor

Auto deletion is enabled.

See also

setAutoDelete(), attachItem()

12.52.2.2 QwtPlotDict::~QwtPlotDict ()

Destructor

If autoDelete is on, all attached items will be deleted

See also

setAutoDelete(), autoDelete(), attachItem()

12.52.3 Member Function Documentation

12.52.3.1 bool QwtPlotDict::autoDelete () const

Returns

true if auto deletion is enabled

See also

setAutoDelete(), attachItem()

12.52.3.2 void QwtPlotDict::detachItems (int *rtti* = `QwtPlotItem::Rtti_PlotItem`, bool *autoDelete* = `true`)

Detach items from the dictionary

Parameters

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

12.52.3.3 const QwtPlotItemList & QwtPlotDict::itemList () const

A QwtPlotItemList of all attached plot items.

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

Returns

List of all attached plot items.

12.52.3.4 void QwtPlotDict::setAutoDelete (bool *autoDelete*)

En/Disable Auto deletion

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

See also

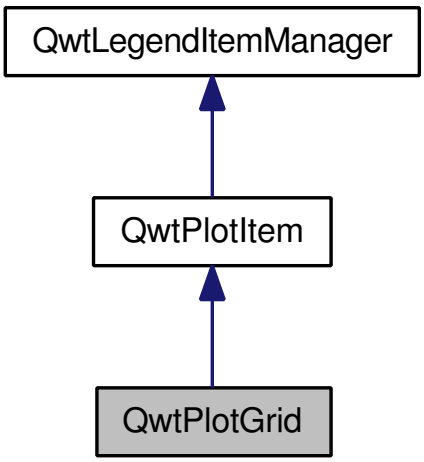
autoDelete(), attachItem()

12.53 QwtPlotGrid Class Reference

A class which draws a coordinate grid.

`#include <qwt_plot_grid.h>`

Inheritance diagram for QwtPlotGrid:



Public Member Functions

- QwtPlotGrid ()
- virtual ~QwtPlotGrid ()
- virtual int rtti () const
- void enableX (bool tf)
- bool xEnabled () const
- void enableY (bool tf)
- bool yEnabled () const
- void enableXMin (bool tf)
- bool xMinEnabled () const
- void enableYMin (bool tf)
- bool yMinEnabled () const
- void setXDiv (const QwtScaleDiv &sx)
- const QwtScaleDiv & xScaleDiv () const
- void setYDiv (const QwtScaleDiv &sy)
- const QwtScaleDiv & yScaleDiv () const
- void setPen (const QPen &p)
- void setMajPen (const QPen &p)
- const QPen & majPen () const
- void setMinPen (const QPen &p)
- const QPen & minPen () const
- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &rect) const
- virtual void updateScaleDiv (const QwtScaleDiv &xMap, const QwtScaleDiv &yMap)

12.53.1 Detailed Description

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

12.53.2 Constructor & Destructor Documentation

12.53.2.1 QwtPlotGrid::QwtPlotGrid () [explicit]

Enables major grid, disables minor grid.

12.53.2.2 QwtPlotGrid::~QwtPlotGrid () [virtual]

Destructor.

12.53.3 Member Function Documentation

12.53.3.1 void QwtPlotGrid::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the grid.

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

Parameters

painter Painter

xMap X axis map

yMap Y axis

canvasRect Contents rect of the plot canvas

Implements QwtPlotItem.

12.53.3.2 void QwtPlotGrid::enableX (bool *tf*)

Enable or disable vertical gridlines.

Parameters

tf Enable (true) or disable

See also

Minor gridlines can be enabled or disabled with enableXMin()

12.53.3.3 void QwtPlotGrid::enableXMin (bool *tf*)

Enable or disable minor vertical gridlines.

Parameters

tf Enable (true) or disable

See also

enableX()

12.53.3.4 void QwtPlotGrid::enableY (bool *tf*)

Enable or disable horizontal gridlines.

Parameters

tf Enable (true) or disable

See also

Minor gridlines can be enabled or disabled with enableYMin()

12.53.3.5 void QwtPlotGrid::enableYMin (bool *tf*)

Enable or disable minor horizontal gridlines.

Parameters

tf Enable (true) or disable

See also

enableY()

12.53.3.6 const QPen & QwtPlotGrid::majPen () const

Returns

the pen for the major gridlines

See also

setMajPen(), setMinPen(), setPen()

12.53.3.7 const QPen & QwtPlotGrid::minPen () const

Returns

the pen for the minor gridlines

See also

setMinPen(), setMajPen(), setPen()

12.53.3.8 int QwtPlotGrid::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotGrid

Reimplemented from QwtPlotItem.

12.53.3.9 void QwtPlotGrid::setMajPen (const QPen & *pen*)

Assign a pen for the major gridlines

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

pen Pen

See also

majPen(), setMinPen(), setPen(), QwtPainter::scaledPen()

12.53.3.10 void QwtPlotGrid::setMinPen (const QPen & *pen*)

Assign a pen for the minor gridlines

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

pen Pen

See also

minPen(), setMajPen(), setPen(), QwtPainter::scaledPen()

12.53.3.11 void QwtPlotGrid::setPen (const QPen & *pen*)

Assign a pen for both major and minor gridlines

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

pen Pen

See also

setMajPen(), setMinPen(), QwtPainter::scaledPen()

12.53.3.12 void QwtPlotGrid::setXDiv (const QwtScaleDiv & *scaleDiv*)

Assign an x axis scale division

Parameters

scaleDiv Scale division

12.53.3.13 void QwtPlotGrid::setYDiv (const QwtScaleDiv & *scaleDiv*)

Assign a y axis division

Parameters

scaleDiv Scale division

12.53.3.14 void QwtPlotGrid::updateScaleDiv (const QwtScaleDiv & *xScaleDiv*, const QwtScaleDiv & *yScaleDiv*) `[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.

12.53.3.15 bool QwtPlotGrid::xEnabled () const

Returns

true if vertical gridlines are enabled

See also

enableX()

12.53.3.16 bool QwtPlotGrid::xMinEnabled () const

Returns

true if minor vertical gridlines are enabled

See also

enableXMin()

12.53.3.17 const QwtScaleDiv & QwtPlotGrid::xScaleDiv () const

Returns

the scale division of the x axis

12.53.3.18 bool QwtPlotGrid::yEnabled () const

Returns

true if horizontal gridlines are enabled

See also

enableY()

12.53.3.19 bool QwtPlotGrid::yMinEnabled () const

Returns

true if minor horizontal gridlines are enabled

See also

enableYMin()

12.53.3.20 const QwtScaleDiv & QwtPlotGrid::yScaleDiv () const

Returns

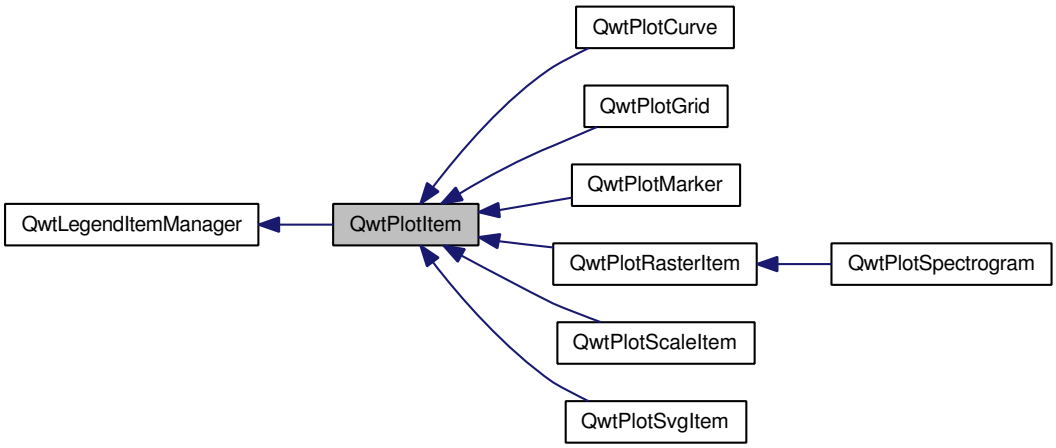
the scale division of the y axis

12.54 QwtPlotItem Class Reference

Base class for items on the plot canvas.

`#include <qwt_plot_item.h>`

Inheritance diagram for QwtPlotItem:



Public Types

- enum RttiValues {
  Rtti_PlotItem = 0,
  Rtti_PlotGrid,
  Rtti_PlotScale,
  Rtti_PlotMarker,
  Rtti_PlotCurve,
  Rtti_PlotHistogram,
  Rtti_PlotSpectrogram,
  Rtti_PlotSVG,
  Rtti_PlotUserItem = 1000 }
- enum ItemAttribute {
  Legend = 1,
  AutoScale = 2 }
- enum RenderHint { **RenderAntialiased** = 1 }

Public Member Functions

- QwtPlotItem (const QwtText &title=QwtText())
- virtual ~QwtPlotItem ()
- void attach (QwtPlot *plot)
- void detach ()
- QwtPlot * 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
- double z () const
- void setZ (double z)
- void show ()
- void hide ()
- virtual void setVisible (bool)
- bool isVisible () const
- void setAxis (int xAxis, int yAxis)
- void setXAxis (int axis)
- int xAxis () const
- void setYAxis (int axis)
- int yAxis () const
- virtual void itemChanged ()
- virtual void draw (QPainter *painter, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &canvasRect) const =0
- virtual QwtDoubleRect boundingRect () const
- virtual void updateLegend (QwtLegend *) const
- virtual void updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &)
- virtual QWidget * legendItem () const
- QwtDoubleRect scaleRect (const QwtScaleMap &, const QwtScaleMap &) const
- QRect paintRect (const QwtScaleMap &, const QwtScaleMap &) const
- QRect transform (const QwtScaleMap &, const QwtScaleMap &, const QwtDoubleRect &) const
- QwtDoubleRect invTransform (const QwtScaleMap &, const QwtScaleMap &, const QRect &) const

12.54.1 Detailed Description

Base class for items on the plot canvas. A plot item is "something", that can be painted on the plot canvas, or only affects the scales of the plot widget. They can be categorized as:

- Representator

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

  - QwtPlotMarker Represents a point or a horizontal/vertical coordinate
  - QwtPlotCurve Represents a series of points
  - QwtPlotSpectrogram (QwtPlotRasterItem) Represents raster data
  - ...

- Decorators

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

  - QwtPlotGrid

- QwtPlotScaleItem
- QwtPlotSvgItem
- ...

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

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

See also

The cpuplot example shows the implementation of additional plot items.

12.54.2 Member Enumeration Documentation

12.54.2.1 enum QwtPlotItem::ItemAttribute

Plot Item Attributes

- Legend
  The item is represented on the legend.
- AutoScale
  The boundingRect() of the item is included in the autoscaling calculation.

See also

setItemAttribute(), testItemAttribute()

12.54.2.2 enum QwtPlotItem::RenderHint

Render hints.

12.54.2.3 enum QwtPlotItem::RttiValues

Runtime type information.

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

12.54.3 Constructor & Destructor Documentation

12.54.3.1 QwtPlotItem::QwtPlotItem (const QwtText & *title* = QwtText ()) [explicit]

Constructor

Parameters

title Title of the item

12.54.3.2 QwtPlotItem::~QwtPlotItem () [virtual]

Destroy the QwtPlotItem.

12.54.4 Member Function Documentation

12.54.4.1 void QwtPlotItem::attach (QwtPlot * *plot*)

Attach the item to a plot.

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

Parameters

plot Plot widget

See also

QwtPlotItem::detach()

12.54.4.2 QwtDoubleRect QwtPlotItem::boundingRect () const [virtual]

Returns

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

Reimplemented in QwtPlotCurve, QwtPlotMarker, QwtPlotSpectrogram, and QwtPlotSvgItem.

12.54.4.3 void QwtPlotItem::detach () [inline]

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

detach() is equivalent to calling attach(NULL)

See also

attach(QwtPlot* plot)

12.54.4.4 virtual void QwtPlotItem::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [pure virtual]

Draw the item.

Parameters

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 QwtPlotCurve, QwtPlotGrid, QwtPlotMarker, QwtPlotRasterItem, QwtPlotScaleItem, QwtPlotSpectrogram, and QwtPlotSvgItem.

12.54.4.5 void QwtPlotItem::hide ()

Hide the item.

12.54.4.6 QwtDoubleRect QwtPlotItem::invTransform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *rect*) const

Transform a rectangle from paint to scale coordinates

Parameters

xMap X map

yMap Y map

rect Rectangle in paint coordinates

Returns

Rectangle in scale coordinates

See also

transform()

12.54.4.7 bool QwtPlotItem::isVisible () const

Returns

true if visible

See also

setVisible(), show(), hide()

12.54.4.8 void QwtPlotItem::itemChanged () `[virtual]`

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

See also

updateLegend()

12.54.4.9 QWidget * QwtPlotItem::legendItem () const [virtual]

Allocate the widget that represents the item on the legend.

The default implementation is made for QwtPlotCurve and returns a QwtLegendItem(), but an item could be represented by any type of widget, by overloading legendItem() and updateLegend().

Returns

QwtLegendItem()

See also

updateLegend() QwtLegend()

Implements QwtLegendItemManager.

12.54.4.10 QRect QwtPlotItem::paintRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const

Calculate the bounding paint rect of 2 maps.

Parameters

xMap X map

yMap X map

Returns

Bounding rect of the scale maps

12.54.4.11 QwtPlot * QwtPlotItem::plot () const

Return attached plot.

12.54.4.12 int QwtPlotItem::rtti () const [virtual]

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

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

Returns

rtti value

See also

RttiValues

Reimplemented in QwtPlotCurve, QwtPlotGrid, QwtPlotMarker, QwtPlotScaleItem, QwtPlotSpectrogram, and QwtPlotSvgItem.

12.54.4.13 QwtDoubleRect QwtPlotItem::scaleRect (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*) const

Calculate the bounding scale rect of 2 maps.

Parameters

xMap X map

yMap X map

Returns

Bounding rect of the scale maps

12.54.4.14 void QwtPlotItem::setAxis (int *xAxis*, int *yAxis*)

Set X and Y axis

The item will painted according to the coordinates its Axes.

Parameters

xAxis X Axis

yAxis Y Axis

See also

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

12.54.4.15 void QwtPlotItem::setItemAttribute (ItemAttribute *attribute*, bool *on* = `true`)

Toggle an item attribute

Parameters

attribute Attribute type

on true/false

See also

testItemAttribute(), ItemAttribute

12.54.4.16 void QwtPlotItem::setRenderHint (RenderHint *hint*, bool *on* = `true`)

Toggle an render hint

Parameters

hint Render hint

on true/false

See also

testRenderHint(), RenderHint

12.54.4.17 void QwtPlotItem::setTitle (const QwtText & *title*)

Set a new title

Parameters

title Title

See also

title()

12.54.4.18 void QwtPlotItem::setTitle (const QString & *title*)

Set a new title

Parameters

title Title

See also

title()

12.54.4.19 void QwtPlotItem::setVisible (bool *on*) `[virtual]`

Show/Hide the item

Parameters

on Show if true, otherwise hide

See also

isVisible(), show(), hide()

12.54.4.20 void QwtPlotItem::setXAxis (int *axis*)

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

axis X Axis

See also

setAxis(), setYAxis(), xAxis()

12.54.4.21 void QwtPlotItem::setYAxis (int *axis*)

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

axis Y Axis

See also

setAxis(), setXAxis(), yAxis()

12.54.4.22 void QwtPlotItem::setZ (double *z*)

Set the z value.

Plot items are painted in increasing z-order.

Parameters

z Z-value

See also

z(), QwtPlotDict::itemList()

12.54.4.23 void QwtPlotItem::show ()

Show the item.

12.54.4.24 bool QwtPlotItem::testItemAttribute (ItemAttribute *attribute*) const

Test an item attribute

Parameters

attribute Attribute type

Returns

true/false

See also

setItemAttribute(), ItemAttribute

12.54.4.25 bool QwtPlotItem::testRenderHint (RenderHint *hint*) const

Test a render hint

Parameters

hint Render hint

Returns

true/false

See also

setRenderHint(), RenderHint

12.54.4.26 const QwtText & QwtPlotItem::title () const

Returns

Title of the item

See also

setTitle()

12.54.4.27 QRect QwtPlotItem::transform (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QwtDoubleRect & *rect*) const

Transform a rectangle

Parameters

xMap X map

yMap Y map

rect Rectangle in scale coordinates

Returns

Rectangle in paint coordinates

See also

invTransform()

12.54.4.28 void QwtPlotItem::updateLegend (QwtLegend * *legend*) const [virtual]

Update the widget that represents the item on the legend.

updateLegend() is called from itemChanged() to adopt the widget representing the item on the legend to its new configuration.

The default implementation is made for QwtPlotCurve and updates a QwtLegendItem(), but an item could be represented by any type of widget, by overloading legendItem() and updateLegend().

Parameters

legend Legend

See also

legendItem(), itemChanged(), QwtLegend()

Implements QwtLegendItemManager.

Reimplemented in QwtPlotCurve.

12.54.4.29 void QwtPlotItem::updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &) [virtual]

Update the item to changes of the axes scale division.

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

Parameters

xScaleDiv Scale division of the x-axis

yScaleDiv Scale division of the y-axis

See also

QwtPlot::updateAxes()

Reimplemented in QwtPlotGrid, and QwtPlotScaleItem.

12.54.4.30 int QwtPlotItem::xAxis () const

Return xAxis.

12.54.4.31 int QwtPlotItem::yAxis () const

Return yAxis.

12.54.4.32 double QwtPlotItem::z () const

Plot items are painted in increasing z-order.

Returns

setZ(), QwtPlotDict::itemList()

12.55 QwtPlotLayout Class Reference

Layout engine for QwtPlot.

`#include <qwt_plot_layout.h>`

Public Types

- enum Options {
  AlignScales = 1,
  IgnoreScrollbars = 2,
  IgnoreFrames = 4,
  IgnoreMargin = 8,
  IgnoreLegend = 16 }

Public Member Functions

- QwtPlotLayout ()
- virtual ~QwtPlotLayout ()
- void setMargin (int)
- int margin () const
- void setCanvasMargin (int margin, int axis=-1)
- int canvasMargin (int axis) const
- void setAlignCanvasToScales (bool)
- bool alignCanvasToScales () const
- void setSpacing (int)
- int spacing () const
- void setLegendPosition (QwtPlot::LegendPosition pos, double ratio)
- void setLegendPosition (QwtPlot::LegendPosition pos)
- QwtPlot::LegendPosition legendPosition () const
- void setLegendRatio (double ratio)
- double legendRatio () const
- virtual QSize minimumSizeHint (const QwtPlot *) const
- virtual void activate (const QwtPlot *, const QRect &rect, int options=0)
- virtual void invalidate ()
- const QRect & titleRect () const
- const QRect & legendRect () const
- const QRect & scaleRect (int axis) const
- const QRect & canvasRect () const

Protected Member Functions

- QRect layoutLegend (int options, const QRect &) const
- QRect alignLegend (const QRect &canvasRect, const QRect &legendRect) const
- void expandLineBreaks (int options, const QRect &rect, int &dimTitle, int dimAxes[QwtPlot::axisCnt]) const
- void alignScales (int options, QRect &canvasRect, QRect scaleRect[QwtPlot::axisCnt]) const

12.55.1 Detailed Description

Layout engine for QwtPlot. It is used by the QwtPlot widget to organize its internal widgets or by QwtPlot::print() to render its content to a QPaintDevice like a QPrinter, QPixmap/QImage or QSvgRenderer.

12.55.2 Member Enumeration Documentation

12.55.2.1 enum QwtPlotLayout::Options

Options to configure the plot layout engine

- AlignScales
  Unused
- IgnoreScrollbars
  Ignore the dimension of the scrollbars. There are no scrollbars, when the plot is rendered to a paint device (QwtPlot::print()).
- IgnoreFrames
  Ignore all frames. QwtPlot::print() doesn't paint them.
- IgnoreMargin
  Ignore the margin().
- IgnoreLegend
  Ignore the legend.

See also

activate()

12.55.3 Constructor & Destructor Documentation

12.55.3.1 QwtPlotLayout::QwtPlotLayout () [explicit]

Constructor.

12.55.3.2 QwtPlotLayout::~QwtPlotLayout () [virtual]

Destructor.

12.55.4 Member Function Documentation

12.55.4.1 void QwtPlotLayout::activate (const QwtPlot * *plot*, const QRect & *plotRect*, int *options* = 0) [virtual]

Recalculate the geometry of all components.

Parameters

plot Plot to be layout

plotRect Rect where to place the components

options Options

See also

invalidate(), Options, titleRect(), legendRect(), scaleRect(), canvasRect()

12.55.4.2 bool QwtPlotLayout::alignCanvasToScales () const

Return the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size
- align with the axis scale ends to control its size.

Returns

align-canvas-to-axis-scales setting

See also

setAlignCanvasToScales, setCanvasMargin()

Note

In this context the term 'scale' means the backbone of a scale.

12.55.4.3 QRect QwtPlotLayout::alignLegend (const QRect & *canvasRect*, const QRect & *legendRect*) const [protected]

Align the legend to the canvas

Parameters

canvasRect Geometry of the canvas

legendRect Maximum geometry for the legend

Returns

Geometry for the aligned legend

12.55.4.4 void QwtPlotLayout::alignScales (int *options*, QRect & *canvasRect*, QRect *scaleRect*[QwtPlot::axisCnt]) const [protected]

Align the ticks of the axis to the canvas borders using the empty corners.

See also

Options

12.55.4.5 int QwtPlotLayout::canvasMargin (int *axis*) const

Returns

Margin around the scale tick borders

See also

setCanvasMargin()

12.55.4.6 const QRect & QwtPlotLayout::canvasRect () const

Returns

Geometry for the canvas

See also

activate(), invalidate()

12.55.4.7 void QwtPlotLayout::expandLineBreaks (int *options*, const QRect & *rect*, int & *dimTitle*, int *dimAxis*[QwtPlot::axisCnt]) const [protected]

Expand all line breaks in text labels, and calculate the height of their widgets in orientation of the text.

Parameters

options Options how to layout the legend

rect Bounding rect for title, axes and canvas.

dimTitle Expanded height of the title widget

dimAxis Expanded heights of the axis in axis orientation.

See also

Options

12.55.4.8 void QwtPlotLayout::invalidate () [virtual]

Invalidate the geometry of all components.

See also

activate()

12.55.4.9 QRect QwtPlotLayout::layoutLegend (int *options*, const QRect & *rect*) const [protected]

Find the geometry for the legend

Parameters

options Options how to layout the legend

rect Rectangle where to place the legend

Returns

Geometry for the legend

See also

Options

12.55.4.10 QwtPlot::LegendPosition QwtPlotLayout::legendPosition () const

Returns

Position of the legend

See also

setLegendPosition(), QwtPlot::setLegendPosition(), QwtPlot::legendPosition()

12.55.4.11 double QwtPlotLayout::legendRatio () const

Returns

The relative size of the legend in the plot.

See also

setLegendPosition()

12.55.4.12 const QRect & QwtPlotLayout::legendRect () const

Returns

Geometry for the legend

See also

activate(), invalidate()

12.55.4.13 int QwtPlotLayout::margin () const

Returns

margin

See also

setMargin(), spacing(), QwtPlot::margin()

12.55.4.14 QSize QwtPlotLayout::minimumSizeHint (const QwtPlot * *plot*) const [virtual]

Return a minimum size hint.

See also

QwtPlot::minimumSizeHint()

12.55.4.15 const QRect & QwtPlotLayout::scaleRect (int *axis*) const

Parameters

axis Axis index

Returns

Geometry for the scale

See also

activate(), invalidate()

12.55.4.16 void QwtPlotLayout::setAlignCanvasToScales (bool *alignCanvasToScales*)

Change the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size,
- align with the axis scale ends to control its size.

Parameters

alignCanvasToScales New align-canvas-to-axis-scales setting

See also

setCanvasMargin()

Note

In this context the term 'scale' means the backbone of a scale.

Warning

In case of alignCanvasToScales == true canvasMargin will have no effect

12.55.4.17 void QwtPlotLayout::setCanvasMargin (int *margin*, int *axis* = -1)

Change a margin of the canvas. The margin is the space above/below the scale ticks. A negative margin will be set to -1, excluding the borders of the scales.

Parameters

margin New margin

axis One of QwtPlot::Axis. Specifies where the position of the margin. -1 means margin at all borders.

See also

canvasMargin()

Warning

The margin will have no effect when alignCanvasToScales is true

12.55.4.18 void QwtPlotLayout::setLegendPosition (QwtPlot::LegendPosition *pos*)

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()

12.55.4.19 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 rect of title, canvas and axes. The legend will be shrinked if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also

QwtPlot::setLegendPosition()

12.55.4.20 void QwtPlotLayout::setLegendRatio (double *ratio*)

Specify the relative size of the legend in the plot

Parameters

ratio Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrinked if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

12.55.4.21 void QwtPlotLayout::setMargin (int *margin*)

Change the margin of the plot. The margin is the space around all components.

Parameters

margin new margin

See also

margin(), setSpacing(), QwtPlot::setMargin()

12.55.4.22 void QwtPlotLayout::setSpacing (int *spacing*)

Change the spacing of the plot. The spacing is the distance between the plot components.

Parameters

spacing new spacing

See also

setMargin(), spacing()

12.55.4.23 int QwtPlotLayout::spacing () const

Returns

spacing

See also

margin(), setSpacing()

12.55.4.24 const QRect & QwtPlotLayout::titleRect () const

Returns

Geometry for the title

See also

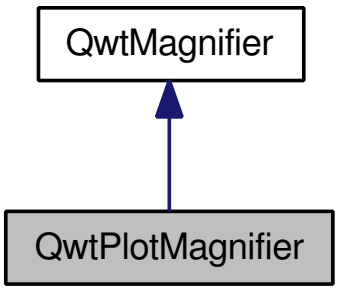
activate(), invalidate()

12.56 QwtPlotMagnifier Class Reference

QwtPlotMagnifier provides zooming, by magnifying in steps.

`#include <qwt_plot_magnifier.h>`

Inheritance diagram for QwtPlotMagnifier:



Public Member Functions

- QwtPlotMagnifier (QwtPlotCanvas ∗)
- virtual ∼QwtPlotMagnifier ()
- void setAxisEnabled (int axis, bool on)
- bool isAxisEnabled (int axis) const
- QwtPlotCanvas ∗ canvas ()
- const QwtPlotCanvas ∗ canvas () const
- QwtPlot ∗ plot ()
- const QwtPlot ∗ plot () const

Protected Member Functions

- virtual void rescale (double factor)

12.56.1 Detailed Description

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

Together with QwtPlotZoomer and QwtPlotPanner it is possible to implement individual and powerful navigation of the plot canvas.

See also

QwtPlotZoomer, QwtPlotPanner, QwtPlot

12.56.2 Constructor & Destructor Documentation

12.56.2.1 QwtPlotMagnifier::QwtPlotMagnifier (QwtPlotCanvas ∗ *canvas*) `[explicit]`

Constructor

Parameters

canvas Plot canvas to be magnified

12.56.2.2 QwtPlotMagnifier::~QwtPlotMagnifier () [virtual]

Destructor.

12.56.3 Member Function Documentation

12.56.3.1 const QwtPlotCanvas * QwtPlotMagnifier::canvas () const

Return Observed plot canvas.

12.56.3.2 QwtPlotCanvas * QwtPlotMagnifier::canvas ()

Return observed plot canvas.

12.56.3.3 bool QwtPlotMagnifier::isAxisEnabled (int *axis*) const

Test if an axis is enabled

Parameters

axis Axis, see QwtPlot::Axis

Returns

True, if the axis is enabled

See also

setAxisEnabled()

12.56.3.4 const QwtPlot * QwtPlotMagnifier::plot () const

Return plot widget, containing the observed plot canvas.

12.56.3.5 QwtPlot * QwtPlotMagnifier::plot ()

Return plot widget, containing the observed plot canvas.

12.56.3.6 void QwtPlotMagnifier::rescale (double *factor*) [protected, virtual]

Zoom in/out the axes scales

Parameters

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

Implements QwtMagnifier.

12.56.3.7 void QwtPlotMagnifier::setAxisEnabled (int *axis*, bool *on*)

En/Disable an axis.

Axes that are enabled will be synchronized to the result of panning. All other axes will remain unchanged.

Parameters

axis Axis, see QwtPlot::Axis

on On/Off

See also

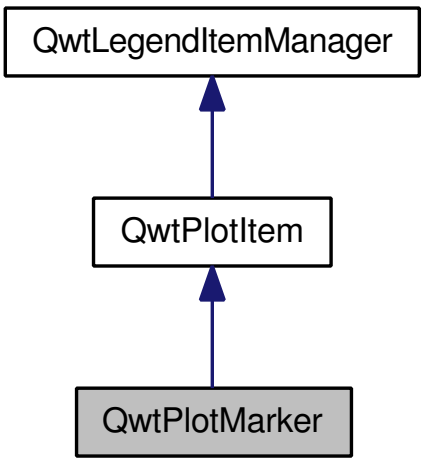
isAxisEnabled()

12.57 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 ()
- virtual ~QwtPlotMarker ()
- virtual int rtti () const
- double xValue () const
- double yValue () const
- QwtDoublePoint value () const
- void setXValue (double)
- void setYValue (double)
- void setValue (double, double)
- void setValue (const QwtDoublePoint &)
- void setLineStyle (LineStyle st)
- LineStyle lineStyle () const
- void setLinePen (const QPen &p)
- const QPen & linePen () const
- void setSymbol (const QwtSymbol &s)
- const QwtSymbol & symbol () const
- void setLabel (const QwtText &)
- QwtText label () const
- void setLabelAlignment (Qt::Alignment)
- Qt::Alignment labelAlignment () const
- void setLabelOrientation (Qt::Orientation)
- Qt::Orientation labelOrientation () const
- void setSpacing (int)
- int spacing () const
- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &) const
- virtual QwtDoubleRect boundingRect () const

Protected Member Functions

- void drawAt (QPainter *, const QRect &, const QPoint &) const

12.57.1 Detailed Description

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

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

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

12.57.2 Member Enumeration Documentation

12.57.2.1 enum QwtPlotMarker::LineStyle

Line styles.

See also

setLineStyle(), lineStyle()

12.57.3 Constructor & Destructor Documentation

12.57.3.1 QwtPlotMarker::QwtPlotMarker () [explicit]

Sets alignment to Qt::AlignCenter, and style to NoLine.

12.57.3.2 QwtPlotMarker::~QwtPlotMarker () [virtual]

Destructor.

12.57.4 Member Function Documentation

12.57.4.1 QwtDoubleRect QwtPlotMarker::boundingRect () const [virtual]

Returns

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

Reimplemented from QwtPlotItem.

12.57.4.2 void QwtPlotMarker::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the marker

Parameters

painter Painter

xMap x Scale Map

yMap y Scale Map

canvasRect Contents rect of the canvas in painter coordinates

Implements QwtPlotItem.

12.57.4.3 void QwtPlotMarker::drawAt (QPainter * *painter*, const QRect & *canvasRect*, const QPoint & *pos*) const [protected]

Draw the marker at a specific position

Parameters

painter Painter

canvasRect Contents rect of the canvas in painter coordinates

pos Position of the marker in painter coordinates

12.57.4.4 QwtText QwtPlotMarker::label () const

Returns

the label

See also

setLabel()

12.57.4.5 Qt::Alignment QwtPlotMarker::labelAlignment () const

Returns

the label alignment

See also

setLabelAlignment(), setLabelOrientation()

12.57.4.6 Qt::Orientation QwtPlotMarker::labelOrientation () const

Returns

the label orientation

See also

setLabelOrientation(), labelAlignment()

12.57.4.7 const QPen & QwtPlotMarker::linePen () const

Returns

the line pen

See also

setLinePen()

12.57.4.8 QwtPlotMarker::LineStyle QwtPlotMarker::lineStyle () const

Returns

the line style

See also

For a description of line styles, see QwtPlotMarker::setLineStyle()

12.57.4.9 int QwtPlotMarker::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotMarker

Reimplemented from QwtPlotItem.

12.57.4.10 void QwtPlotMarker::setLabel (const QwtText & *label*)

Set the label.

Parameters

label label text

See also

label()

12.57.4.11 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. A combination of AlignTop, AlignBottom, AlignLeft, AlignRight, AlignCenter, AlgnHCenter, AlignVCenter.

See also

labelAlignment(), labelOrientation()

12.57.4.12 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()

12.57.4.13 void QwtPlotMarker::setLinePen (const QPen & *pen*)

Specify a pen for the line.

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

pen New pen

See also

linePen(), QwtPainter::scaledPen()

12.57.4.14 void QwtPlotMarker::setLineStyle (QwtPlotMarker::LineStyle *st*)

Set the line style.

Parameters

st Line style. Can be one of QwtPlotMarker::NoLine, HLine, VLine or Cross

See also

lineStyle()

12.57.4.15 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()

12.57.4.16 void QwtPlotMarker::setSymbol (const QwtSymbol & *s*)

Assign a symbol.

Parameters

s New symbol

See also

symbol()

12.57.4.17 void QwtPlotMarker::setValue (const QwtDoublePoint & *pos*)

Set Value.

12.57.4.18 void QwtPlotMarker::setValue (double *x*, double *y*)

Set Value.

12.57.4.19 void QwtPlotMarker::setXValue (double *x*)

Set X Value.

12.57.4.20 void QwtPlotMarker::setYValue (double *y*)

Set Y Value.

12.57.4.21 int QwtPlotMarker::spacing () const

Returns

the spacing

See also

setSpacing()

12.57.4.22 const QwtSymbol & QwtPlotMarker::symbol () const

Returns

the symbol

See also

setSymbol(), QwtSymbol

12.57.4.23 QwtDoublePoint QwtPlotMarker::value () const

Return Value.

12.57.4.24 double QwtPlotMarker::xValue () const

Return x Value.

12.57.4.25 double QwtPlotMarker::yValue () const

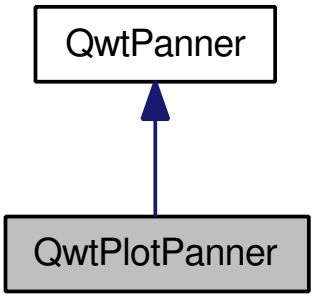
Return y Value.

12.58 QwtPlotPanner Class Reference

QwtPlotPanner provides panning of a plot canvas.

```
#include <qwt_plot_panner.h>
```

Inheritance diagram for QwtPlotPanner:



Public Member Functions

- QwtPlotPanner (QwtPlotCanvas ∗)
- virtual ∼QwtPlotPanner ()
- QwtPlotCanvas ∗ canvas ()
- const QwtPlotCanvas ∗ canvas () const
- QwtPlot ∗ plot ()
- const QwtPlot ∗ plot () const
- void setAxisEnabled (int axis, bool on)
- bool isAxisEnabled (int axis) const

Protected Slots

- virtual void moveCanvas (int dx, int dy)

12.58.1 Detailed Description

QwtPlotPanner provides panning of a plot canvas. QwtPlotPanner is a panner for a QwtPlotCanvas, that adjusts the scales of the axes after dropping the canvas on its new position.

Together with QwtPlotZoomer and QwtPlotMagnifier powerful ways of navigating on a QwtPlot widget can be implemented easily.

Note

The axes are not updated, while dragging the canvas

See also

QwtPlotZoomer, QwtPlotMagnifier

12.58.2 Constructor & Destructor Documentation

12.58.2.1 QwtPlotPanner::QwtPlotPanner (QwtPlotCanvas ∗ *canvas*) `[explicit]`

Create a plot panner.

The panner is enabled for all axes

Parameters

canvas Plot canvas to pan, also the parent object

See also

setAxisEnabled()

12.58.2.2 QwtPlotPanner::~QwtPlotPanner () [virtual]

Destructor.

12.58.3 Member Function Documentation

12.58.3.1 const QwtPlotCanvas * QwtPlotPanner::canvas () const

Return Observed plot canvas.

12.58.3.2 QwtPlotCanvas * QwtPlotPanner::canvas ()

Return observed plot canvas.

12.58.3.3 bool QwtPlotPanner::isAxisEnabled (int *axis*) const

Test if an axis is enabled

Parameters

axis Axis, see QwtPlot::Axis

Returns

True, if the axis is enabled

See also

setAxisEnabled(), moveCanvas()

12.58.3.4 void QwtPlotPanner::moveCanvas (int *dx*, int *dy*) [protected, 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()

12.58.3.5 const QwtPlot * QwtPlotPanner::plot () const

Return plot widget, containing the observed plot canvas.

12.58.3.6 QwtPlot * QwtPlotPanner::plot ()

Return plot widget, containing the observed plot canvas.

12.58.3.7 void QwtPlotPanner::setAxisEnabled (int *axis*, bool *on*)

En/Disable an axis.

Axes that are enabled will be synchronized to the result of panning. All other axes will remain unchanged.

Parameters

axis Axis, see QwtPlot::Axis

on On/Off

See also

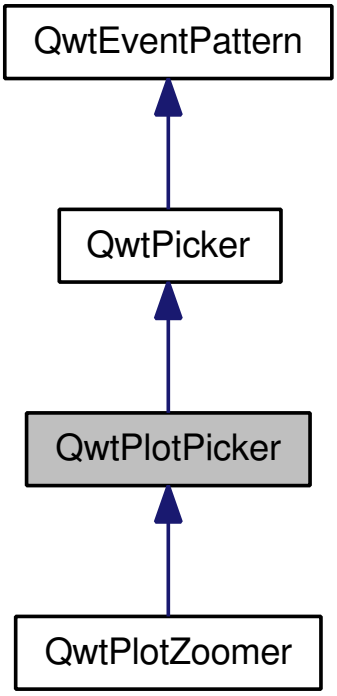
isAxisEnabled(), moveCanvas()

12.59 QwtPlotPicker Class Reference

QwtPlotPicker provides selections on a plot canvas.

```
#include <qwt_plot_picker.h>
```

Inheritance diagram for QwtPlotPicker:



Signals

- void selected (const QwtDoublePoint &pos)
- void selected (const QwtDoubleRect &rect)
- void selected (const QwtArray< QwtDoublePoint > &pa)
- void appended (const QwtDoublePoint &pos)
- void moved (const QwtDoublePoint &pos)

Public Member Functions

- QwtPlotPicker (QwtPlotCanvas *)
- virtual ~QwtPlotPicker ()
- QwtPlotPicker (int xAxis, int yAxis, QwtPlotCanvas *)
- QwtPlotPicker (int xAxis, int yAxis, int selectionFlags, RubberBand rubberBand, DisplayMode trackerMode, QwtPlotCanvas *)
- virtual void setAxis (int xAxis, int yAxis)
- int xAxis () const
- int yAxis () const
- QwtPlot * plot ()
- const QwtPlot * plot () const
- QwtPlotCanvas * canvas ()
- const QwtPlotCanvas * canvas () const

Protected Member Functions

- QwtDoubleRect scaleRect () const
- QwtDoubleRect invTransform (const QRect &) const
- QRect transform (const QwtDoubleRect &) const
- QwtDoublePoint invTransform (const QPoint &) const
- QPoint transform (const QwtDoublePoint &) const
- virtual QwtText trackerText (const QPoint &) const
- virtual QwtText trackerText (const QwtDoublePoint &) const
- virtual void move (const QPoint &)
- virtual void append (const QPoint &)
- virtual bool end (bool ok=true)

12.59.1 Detailed Description

QwtPlotPicker provides selections on a plot canvas. QwtPlotPicker is a QwtPicker tailored for selections on a plot canvas. It is set to a x-Axis and y-Axis and translates all pixel coordinates into this coodinate system.

12.59.2 Constructor & Destructor Documentation

12.59.2.1 QwtPlotPicker::QwtPlotPicker (QwtPlotCanvas * *canvas*) [explicit]

Create a plot picker.

The picker is set to those x- and y-axis of the plot that are enabled. If both or no x-axis are enabled, the picker is set to QwtPlot::xBottom. If both or no y-axis are enabled, it is set to QwtPlot::yLeft.

Parameters

canvas Plot canvas to observe, also the parent object

See also

QwtPlot::autoReplot(), QwtPlot::replot(), QwtPlotPicker::scaleRect()

12.59.2.2 QwtPlotPicker::~QwtPlotPicker () [virtual]

Destructor.

12.59.2.3 QwtPlotPicker::QwtPlotPicker (int *xAxis*, int *yAxis*, QwtPlotCanvas * *canvas*) [explicit]

Create a plot picker

Parameters

xAxis Set the x axis of the picker

yAxis Set the y axis of the picker

canvas Plot canvas to observe, also the parent object

See also

QwtPlot::autoReplot(), QwtPlot::replot(), QwtPlotPicker::scaleRect()

12.59.2.4 QwtPlotPicker::QwtPlotPicker (int *xAxis*, int *yAxis*, int *selectionFlags*, RubberBand *rubberBand*, DisplayMode *trackerMode*, QwtPlotCanvas * *canvas*) [explicit]

Create a plot picker

Parameters

xAxis X axis of the picker

yAxis Y axis of the picker

selectionFlags Or'd value of SelectionType, RectSelectionType and SelectionMode

rubberBand Rubberband 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(), QwtPlotPicker::scaleRect()

12.59.3 Member Function Documentation

12.59.3.1 void QwtPlotPicker::append (const QPoint & *pos*) [protected, virtual]

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

Parameters

pos Additional point

See also

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

Note

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

Reimplemented from QwtPicker.

12.59.3.2 void QwtPlotPicker::appended (const QwtDoublePoint & *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()

12.59.3.3 const QwtPlotCanvas * QwtPlotPicker::canvas () const

Return Observed plot canvas.

12.59.3.4 QwtPlotCanvas * QwtPlotPicker::canvas ()

Return observed plot canvas.

12.59.3.5 bool QwtPlotPicker::end (bool *ok* = `true`) `[protected, virtual]`

Close a selection setting the state to inactive.

Parameters

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

Returns

true if the selection is accepted, false otherwise

Reimplemented from QwtPicker.

Reimplemented in QwtPlotZoomer.

12.59.3.6 QwtDoublePoint QwtPlotPicker::invTransform (const QPoint & *pos*) const `[protected]`

Translate a point from pixel into plot coordinates

Returns

Point in plot coordinates

See also

QwtPlotPicker::transform()

12.59.3.7 QwtDoubleRect QwtPlotPicker::invTransform (const QRect & *rect*) const `[protected]`

Translate a rectangle from pixel into plot coordinates

Returns

Rectangle in plot coordinates

See also

QwtPlotPicker::transform()

12.59.3.8 void QwtPlotPicker::move (const QPoint & *pos*) [protected, virtual]

Move the last point of the selection

Parameters

pos New position

See also

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

Note

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

Reimplemented from QwtPicker.

12.59.3.9 void QwtPlotPicker::moved (const QwtDoublePoint & *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()

12.59.3.10 const QwtPlot * QwtPlotPicker::plot () const

Return plot widget, containing the observed plot canvas.

12.59.3.11 QwtPlot * QwtPlotPicker::plot ()

Return plot widget, containing the observed plot canvas.

12.59.3.12 QwtDoubleRect QwtPlotPicker::scaleRect () const [protected]

Return normalized bounding rect of the axes

See also

QwtPlot::autoReplot(), QwtPlot::replot().

12.59.3.13 void QwtPlotPicker::selected (const QwtArray< QwtDoublePoint > & *pa*) [signal]

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

Parameters

pa Selected points

12.59.3.14 void QwtPlotPicker::selected (const QwtDoubleRect & *rect*) [signal]

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

Parameters

rect Selected rectangle

12.59.3.15 void QwtPlotPicker::selected (const QwtDoublePoint & *pos*) [signal]

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

Parameters

pos Selected point

12.59.3.16 void QwtPlotPicker::setAxis (int *xAxis*, int *yAxis*) [virtual]

Set the x and y axes of the picker

Parameters

xAxis X axis

yAxis Y axis

Reimplemented in QwtPlotZoomer.

12.59.3.17 QwtText QwtPlotPicker::trackerText (const QwtDoublePoint & *pos*) const [protected, virtual]

Translate a position into a position string.

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

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

Parameters

pos Position

Returns

Position string

12.59.3.18 QwtText QwtPlotPicker::trackerText (const QPoint & *pos*) const [protected, virtual]

Translate a pixel position into a position string

Parameters

pos Position in pixel coordinates

Returns

Position string

Reimplemented from QwtPicker.

12.59.3.19 QPoint QwtPlotPicker::transform (const QwtDoublePoint & *pos*) const [protected]

Translate a point from plot into pixel coordinates

Returns

Point in pixel coordinates

See also

QwtPlotPicker::invTransform()

12.59.3.20 QRect QwtPlotPicker::transform (const QwtDoubleRect & *rect*) const [protected]

Translate a rectangle from plot into pixel coordinates

Returns

Rectangle in pixel coordinates

See also

QwtPlotPicker::invTransform()

12.59.3.21 int QwtPlotPicker::xAxis () const

Return x axis.

12.59.3.22 int QwtPlotPicker::yAxis () const

Return y axis.

12.60 QwtPlotPrintFilter Class Reference

A base class for plot print filters.

`#include <qwt_plot_printfilter.h>`

Public Types

- enum Options {
  PrintMargin = 1,
  PrintTitle = 2,
  PrintLegend = 4,
  PrintGrid = 8,
  PrintBackground = 16,
  PrintFrameWithScales = 32,
  PrintAll = ~PrintFrameWithScales }
- enum Item {
  Title,
  Legend,
  Curve,
  CurveSymbol,
  Marker,
  MarkerSymbol,
  MajorGrid,
  MinorGrid,
  CanvasBackground,
  AxisScale,
  AxisTitle,
  WidgetBackground }

Public Member Functions

- QwtPlotPrintFilter ()
- virtual ~QwtPlotPrintFilter ()
- virtual QColor color (const QColor &, Item item) const
- virtual QFont font (const QFont &, Item item) const
- void setOptions (int options)
- int options () const
- virtual void apply (QwtPlot *) const
- virtual void reset (QwtPlot *) const
- virtual void **apply** (QwtPlotItem *) const
- virtual void **reset** (QwtPlotItem *) const

12.60.1 Detailed Description

A base class for plot print filters. A print filter can be used to customize QwtPlot::print().

Deprecated

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

12.60.2 Member Enumeration Documentation

12.60.2.1 enum QwtPlotPrintFilter::Item

Print items.

12.60.2.2 enum QwtPlotPrintFilter::Options

Print options.

12.60.3 Constructor & Destructor Documentation

12.60.3.1 QwtPlotPrintFilter::QwtPlotPrintFilter () `[explicit]`

Sets filter options to PrintAll

12.60.3.2 QwtPlotPrintFilter::~QwtPlotPrintFilter () `[virtual]`

Destructor.

12.60.4 Member Function Documentation

12.60.4.1 void QwtPlotPrintFilter::apply (QwtPlot * *plot*) const [virtual]

Change color and fonts of a plot

See also

apply()

12.60.4.2 QColor QwtPlotPrintFilter::color (const QColor & *c*, Item *item*) const [virtual]

Modifies a color for printing.

Parameters

c Color to be modified

item Type of item where the color belongs

Returns

Modified color.

In case of !(QwtPlotPrintFilter::options() & PrintBackground) MajorGrid is modified to Qt::darkGray, MinorGrid to Qt::gray. All other colors are returned unmodified.

12.60.4.3 QFont QwtPlotPrintFilter::font (const QFont & *f*, Item *item*) const [virtual]

Modifies a font for printing.

Parameters

f Font to be modified

item Type of item where the font belongs

All fonts are returned unmodified

12.60.4.4 int QwtPlotPrintFilter::options () const

Get plot print options.

See also

setOptions()

12.60.4.5 void QwtPlotPrintFilter::reset (QwtPlot * *plot*) const [virtual]

Reset color and fonts of a plot

See also

apply()

12.60.4.6 void QwtPlotPrintFilter::setOptions (int *options*)

Set plot print options.

Parameters

options Or'd QwtPlotPrintFilter::Options values

See also

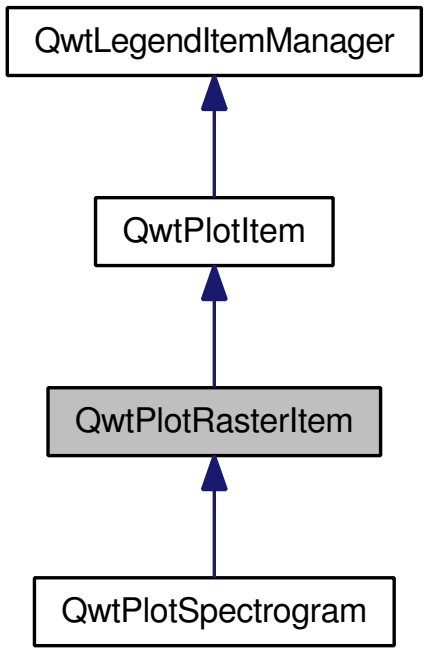
options()

12.61 QwtPlotRasterItem Class Reference

A class, which displays raster data.

`#include <qwt_plot_rasteritem.h>`

Inheritance diagram for QwtPlotRasterItem:



Public Types

- enum CachePolicy {
  NoCache,
  PaintCache,
  ScreenCache }

Public Member Functions

- QwtPlotRasterItem (const QString &title=QString::null)
- QwtPlotRasterItem (const QwtText &title)
- virtual ~QwtPlotRasterItem ()
- void setAlpha (int alpha)
- int alpha () const
- void setCachePolicy (CachePolicy)
- CachePolicy cachePolicy () const
- void invalidateCache ()
- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &rect) const
- virtual QSize rasterHint (const QwtDoubleRect &) const

Protected Member Functions

- virtual QImage renderImage (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtDoubleRect &area) const =0

12.61.1 Detailed Description

A class, which displays raster data. Raster data is a grid of pixel values, that can be represented as a QImage. It is used for many types of information like spectrograms, cartograms, geographical maps ...

Often a plot has several types of raster data organized in layers. (f.e a geographical map, with weather statistics). Using setAlpha() raster items can be stacked easily.

QwtPlotRasterItem is only implemented for images of the following formats: QImage::Format_Indexed8, QImage::Format_ARGB32.

See also

QwtPlotSpectrogram

12.61.2 Member Enumeration Documentation

12.61.2.1 enum QwtPlotRasterItem::CachePolicy

- NoCache
  renderImage() is called, whenever the item has to be repainted

- PaintCache

  renderImage() is called, whenever the image cache is not valid, or the scales, or the size of the canvas has changed. This type of cache is only useful for improving the performance of hide/show operations. All other situations are already handled by the plot canvas cache.

- ScreenCache

  The screen cache is an image in size of the screen. As long as the scales don't change the target image is scaled from the cache. This might improve the performance when resizing the plot widget, but suffers from scaling effects.

The default policy is NoCache

12.61.3 Constructor & Destructor Documentation

12.61.3.1 QwtPlotRasterItem::QwtPlotRasterItem (const QString & *title* = `QString::null`) `[explicit]`

Constructor.

12.61.3.2 QwtPlotRasterItem::QwtPlotRasterItem (const QwtText & *title*) `[explicit]`

Constructor.

12.61.3.3 QwtPlotRasterItem::~QwtPlotRasterItem () `[virtual]`

Destructor.

12.61.4 Member Function Documentation

12.61.4.1 int QwtPlotRasterItem::alpha () const

Returns

Alpha value of the raster item

See also

setAlpha()

12.61.4.2 QwtPlotRasterItem::CachePolicy QwtPlotRasterItem::cachePolicy () const

Returns

Cache policy

See also

CachePolicy, setCachePolicy()

12.61.4.3 void QwtPlotRasterItem::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the raster data.

Parameters

painter Painter

xMap X-Scale Map

yMap Y-Scale Map

canvasRect Contents rect of the plot canvas

Implements QwtPlotItem.

Reimplemented in QwtPlotSpectrogram.

12.61.4.4 void QwtPlotRasterItem::invalidateCache ()

Invalidate the paint cache

See also

setCachePolicy()

12.61.4.5 QSize QwtPlotRasterItem::rasterHint (const QwtDoubleRect &) const [virtual]

Returns the recommended raster for a given rect.

F.e the raster hint can be used to limit the resolution of the image that is rendered.

The default implementation returns an invalid size (QSize()), what means: no hint.

Reimplemented in QwtPlotSpectrogram.

12.61.4.6 virtual QImage QwtPlotRasterItem::renderImage (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QwtDoubleRect & *area*) const [protected, pure virtual]

Renders an image for an area

The format of the image must be QImage::Format_Indexed8, QImage::Format_RGB32 or QImage::Format_ARGB32

Parameters

xMap Maps x-values into pixel coordinates.

yMap Maps y-values into pixel coordinates.

area Requested area for the image in scale coordinates

Implemented in QwtPlotSpectrogram.

12.61.4.7 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()

12.61.4.8 void QwtPlotRasterItem::setCachePolicy (QwtPlotRasterItem::CachePolicy *policy*)

Change the cache policy

The default policy is NoCache

Parameters

policy Cache policy

See also

CachePolicy, cachePolicy()

12.62 QwtPlotRescaler Class Reference

QwtPlotRescaler takes care of fixed aspect ratios for plot scales.

```
#include <qwt_plot_rescaler.h>
```

Public Types

- enum RescalePolicy {
  Fixed,
  Expanding,
  Fitting }
- enum **ExpandingDirection** {
  ExpandUp,
  ExpandDown,
  ExpandBoth }

Public Member Functions

- QwtPlotRescaler (QwtPlotCanvas ∗, int referenceAxis=QwtPlot::xBottom, RescalePolicy=Expanding)
- virtual ∼QwtPlotRescaler ()
- void setEnabled (bool)
- bool isEnabled () const
- void setRescalePolicy (RescalePolicy)
- RescalePolicy rescalePolicy () const
- void setExpandingDirection (ExpandingDirection)
- void setExpandingDirection (int axis, ExpandingDirection)
- ExpandingDirection expandingDirection (int axis) const
- void setReferenceAxis (int axis)
- int referenceAxis () const
- void setAspectRatio (double ratio)
- void setAspectRatio (int axis, double ratio)
- double aspectRatio (int axis) const
- void **setIntervalHint** (int axis, const QwtDoubleInterval &)
- QwtDoubleInterval **intervalHint** (int axis) const
- QwtPlotCanvas ∗ canvas ()
- const QwtPlotCanvas ∗ canvas () const
- QwtPlot ∗ plot ()
- const QwtPlot ∗ plot () const
- virtual bool eventFilter (QObject ∗, QEvent ∗)
- void rescale () const

Protected Member Functions

- virtual void **canvasResizeEvent** (QResizeEvent ∗)
- virtual void rescale (const QSize &oldSize, const QSize &newSize) const
- virtual QwtDoubleInterval expandScale (int axis, const QSize &oldSize, const QSize &newSize) const
- virtual QwtDoubleInterval syncScale (int axis, const QwtDoubleInterval &reference, const QSize &size) const
- virtual void updateScales (QwtDoubleInterval intervals[QwtPlot::axisCnt]) const
- Qt::Orientation orientation (int axis) const
- QwtDoubleInterval interval (int axis) const
- QwtDoubleInterval expandInterval (const QwtDoubleInterval &, double width, ExpandingDirection) const

12.62.1 Detailed Description

QwtPlotRescaler takes care of fixed aspect ratios for plot scales. QwtPlotRescaler autoadjusts the axes of a QwtPlot according to fixed aspect ratios.

12.62.2 Member Enumeration Documentation

12.62.2.1 enum QwtPlotRescaler::RescalePolicy

Rescale Policy.

The rescale policy defines how to rescale the reference axis and their depending axes.

- Fixed

The interval of the reference axis remains unchanged, when the geometry of the canvas changes. All other axes will be adjusted according to their aspect ratio.

- Expanding

The interval of the reference axis will be shrinked/expanded, when the geometry of the canvas changes. All other axes will be adjusted according to their aspect ratio.

The interval, that is represented by one pixel is fixed.

- Fitting

The intervals of the axes are calculated, so that all axes include their minimal interval.

12.62.3 Constructor & Destructor Documentation

12.62.3.1 QwtPlotRescaler::QwtPlotRescaler (QwtPlotCanvas * *canvas*, int *referenceAxis* = `QwtPlot::xBottom`, RescalePolicy *policy* = `Expanding`) `[explicit]`

Constructor

Parameters

canvas Canvas

referenceAxis Reference axis, see RescalePolicy

policy Rescale policy

See also

setRescalePolicy(), setReferenceAxis()

12.62.3.2 QwtPlotRescaler::~QwtPlotRescaler () `[virtual]`

Destructor.

12.62.4 Member Function Documentation

12.62.4.1 double QwtPlotRescaler::aspectRatio (int *axis*) const

Return aspect ratio between an axis and the reference axis.

Parameters

axis Axis index (see QwtPlot::AxisId)

See also

setAspectRatio()

12.62.4.2 const QwtPlotCanvas * QwtPlotRescaler::canvas () const

Returns

plot canvas

12.62.4.3 QwtPlotCanvas * QwtPlotRescaler::canvas ()

Returns

plot canvas

12.62.4.4 bool QwtPlotRescaler::eventFilter (QObject * *o*, QEvent * *e*) [virtual]

Event filter for the plot canvas.

12.62.4.5 QwtPlotRescaler::ExpandingDirection QwtPlotRescaler::expandingDirection (int *axis*) const

Return direction in which an axis should be expanded

Parameters

axis Axis index (see QwtPlot::AxisId)

See also

setExpandingDirection()

12.62.4.6 QwtDoubleInterval QwtPlotRescaler::expandInterval (const QwtDoubleInterval & *interval*, double *width*, ExpandingDirection *direction*) const [protected]

Expand the interval

Parameters

interval Interval to be expanded

width Distance to be added to the interval

direction Direction of the expand operation

Returns

Expanded interval

12.62.4.7 QwtDoubleInterval QwtPlotRescaler::expandScale (int *axis*, const QSize & *oldSize*, const QSize & *newSize*) const [protected, virtual]

Calculate the new scale interval of a plot axis

Parameters

axis Axis index (see QwtPlot::AxisId)

oldSize Previous size of the canvas

newSize New size of the canvas

Returns

Calculated new interval for the axis

12.62.4.8 QwtDoubleInterval QwtPlotRescaler::interval (int *axis*) const [protected]

Return interval of an axis

Parameters

axis Axis index (see QwtPlot::AxisId)

12.62.4.9 bool QwtPlotRescaler::isEnabled () const

Returns

true when enabled, false otherwise

See also

setEnabled, eventFilter()

12.62.4.10 Qt::Orientation QwtPlotRescaler::orientation (int *axis*) const [protected]

Return orientation of an axis

Parameters

axis Axis index (see QwtPlot::AxisId)

12.62.4.11 const QwtPlot * QwtPlotRescaler::plot () const

Returns

plot widget

12.62.4.12 QwtPlot * QwtPlotRescaler::plot ()

Returns

plot widget

12.62.4.13 int QwtPlotRescaler::referenceAxis () const

Returns

Reference axis (see RescalePolicy)

See also

setReferenceAxis()

12.62.4.14 void QwtPlotRescaler::rescale (const 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

12.62.4.15 void QwtPlotRescaler::rescale () const

Adjust the plot axes scales.

12.62.4.16 QwtPlotRescaler::RescalePolicy QwtPlotRescaler::rescalePolicy () const

Returns

Rescale policy

See also

setRescalePolicy()

12.62.4.17 void QwtPlotRescaler::setAspectRatio (int *axis*, double *ratio*)

Set the aspect ratio between the scale of the reference axis and another scale. The default ratio is 1.0

Parameters

axis Axis index (see QwtPlot::AxisId)

ratio Aspect ratio

See also

aspectRatio()

12.62.4.18 void QwtPlotRescaler::setAspectRatio (double *ratio*)

Set the aspect ratio between the scale of the reference axis and the other scales. The default ratio is 1.0

Parameters

ratio Aspect ratio

See also

aspectRatio()

12.62.4.19 void QwtPlotRescaler::setEnabled (bool *on*)

En/disable the rescaler.

When enabled is true an event filter is installed for the canvas, otherwise the event filter is removed.

Parameters

on true or false

See also

isEnabled(), eventFilter()

12.62.4.20 void QwtPlotRescaler::setExpandingDirection (int *axis*, ExpandingDirection *direction*)

Set the direction in which an axis should be expanded

Parameters

axis Axis index (see QwtPlot::AxisId)

direction Direction

See also

expandingDirection()

12.62.4.21 void QwtPlotRescaler::setExpandingDirection (ExpandingDirection *direction*)

Set the direction in which all axis should be expanded

Parameters

direction Direction

See also

expandingDirection()

12.62.4.22 void QwtPlotRescaler::setReferenceAxis (int *axis*)

Set the reference axis (see RescalePolicy)

Parameters

axis Axis index (QwtPlot::Axis)

See also

referenceAxis()

12.62.4.23 void QwtPlotRescaler::setRescalePolicy (RescalePolicy *policy*)

Change the rescale policy

Parameters

policy Rescale policy

See also

rescalePolicy()

12.62.4.24 QwtDoubleInterval QwtPlotRescaler::syncScale (int *axis*, const QwtDoubleInterval & *reference*, const QSize & *size*) const [protected, virtual]

Synchronize an axis scale according to the scale of the reference axis

Parameters

axis Axis index (see QwtPlot::AxisId)

reference Interval of the reference axis

size Size of the canvas

12.62.4.25 void QwtPlotRescaler::updateScales (QwtDoubleInterval *intervals*[QwtPlot::axisCnt]) const [protected, virtual]

Update the axes scales

Parameters

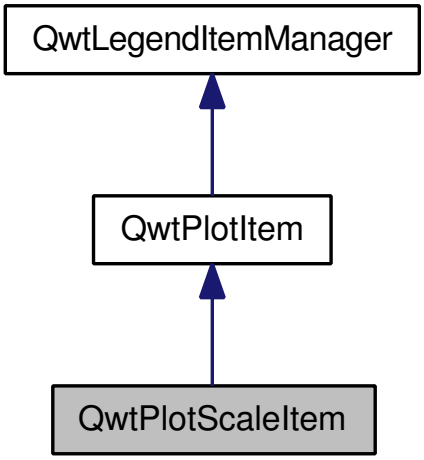
intervals Scale intervals

12.63 QwtPlotScaleItem Class Reference

A class which draws a scale inside the plot canvas.

`#include <qwt_plot_scaleitem.h>`

Inheritance diagram for QwtPlotScaleItem:



Public Member Functions

- QwtPlotScaleItem (QwtScaleDraw::Alignment=QwtScaleDraw::BottomScale, const double pos=0.0)
- virtual ~QwtPlotScaleItem ()
- virtual int rtti () const
- void setScaleDiv (const QwtScaleDiv &)

- 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 ∗)
- const QwtScaleDraw ∗ scaleDraw () const
- QwtScaleDraw ∗ scaleDraw ()
- void setPosition (double pos)
- double position () const
- void setBorderDistance (int numPixels)
- int borderDistance () const
- void setAlignment (QwtScaleDraw::Alignment)
- virtual void draw (QPainter ∗p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &rect) const
- virtual void updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &)

12.63.1 Detailed Description

A class which draws a scale inside the plot canvas. QwtPlotScaleItem can be used to draw an axis inside the plot canvas. It might 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(QwtPlot::yLeft)->font());
scaleItem->attach(plot);

plot->enableAxis(QwtPlot::yLeft, false);
```

12.63.2 Constructor & Destructor Documentation

12.63.2.1 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/QwtScaleDrawTopScale 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()

12.63.2.2 QwtPlotScaleItem::~QwtPlotScaleItem () [virtual]

Destructor.

12.63.3 Member Function Documentation

12.63.3.1 int QwtPlotScaleItem::borderDistance () const

Returns

Distance from a canvas border

See also

setBorderDistance(), setPosition()

12.63.3.2 void QwtPlotScaleItem::draw (QPainter * *p*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *rect*) const [virtual]

Draw the scale.

Implements QwtPlotItem.

12.63.3.3 QFont QwtPlotScaleItem::font () const

Returns

tick label font

See also

setFont()

12.63.3.4 bool QwtPlotScaleItem::isScaleDivFromAxis () const

Returns

True, if the synchronization of the scale division with the corresponding axis is enabled.

See also

setScaleDiv(), setScaleDivFromAxis()

12.63.3.5 QPalette QwtPlotScaleItem::palette () const

Returns

palette

See also

setPalette()

12.63.3.6 double QwtPlotScaleItem::position () const

Returns

Position of the scale

See also

setPosition(), setAlignment()

12.63.3.7 int QwtPlotScaleItem::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotScale

Reimplemented from QwtPlotItem.

12.63.3.8 const QwtScaleDiv & QwtPlotScaleItem::scaleDiv () const

Returns

Scale division

12.63.3.9 QwtScaleDraw * QwtPlotScaleItem::scaleDraw ()

Returns

Scale draw

See also

setScaleDraw()

12.63.3.10 const QwtScaleDraw * QwtPlotScaleItem::scaleDraw () const

Returns

Scale draw

See also

setScaleDraw()

12.63.3.11 void QwtPlotScaleItem::setAlignment (QwtScaleDraw::Alignment *alignment*)

Change the alignment of the scale

The alignment sets the orientation of the scale and the position of the ticks:

- QwtScaleDraw::BottomScale: horizontal, ticks below
- QwtScaleDraw::TopScale: horizontal, ticks above
- QwtScaleDraw::LeftScale: vertical, ticks left
- QwtScaleDraw::RightScale: vertical, ticks right

For horizontal scales the position corresponds to QwtPlotItem::yAxis(), otherwise to QwtPlotItem::xAxis().

See also

scaleDraw(), QwtScaleDraw::alignment(), setPosition()

12.63.3.12 void QwtPlotScaleItem::setBorderDistance (int *distance*)

Align the scale to the canvas.

If distance is >= 0 the scale will be aligned to a border of the contents rect of the canvas. If alignment() is QwtScaleDraw::LeftScale, the scale will be aligned to the right border, if it is QwtScaleDraw::TopScale it will be aligned to the bottom (and vice versa),

If distance is < 0 the scale will be at the position().

Parameters

distance Number of pixels between the canvas border and the backbone of the scale.

See also

setPosition(), borderDistance()

12.63.3.13 void QwtPlotScaleItem::setFont (const QFont & *font*)

Change the tick label font

See also

font()

12.63.3.14 void QwtPlotScaleItem::setPalette (const QPalette & *palette*)

Set the palette

See also

QwtAbstractScaleDraw::draw(), palette()

12.63.3.15 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()

12.63.3.16 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()

12.63.3.17 void QwtPlotScaleItem::setScaleDivFromAxis (bool *on*)

Enable/Disable the synchronization of the scale division with the corresponding axis.

Parameters

on true/false

See also

isScaleDivFromAxis()

12.63.3.18 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()

12.63.3.19 void QwtPlotScaleItem::updateScaleDiv (const QwtScaleDiv & *xScaleDiv*, const QwtScaleDiv & *yScaleDiv*) [virtual]

Update the item to changes of the axes scale division.

In case of isScaleDivFromAxis(), the scale draw is synchronized to the correspond axis.

Parameters

xScaleDiv Scale division of the x-axis

yScaleDiv Scale division of the y-axis

See also

QwtPlot::updateAxes()

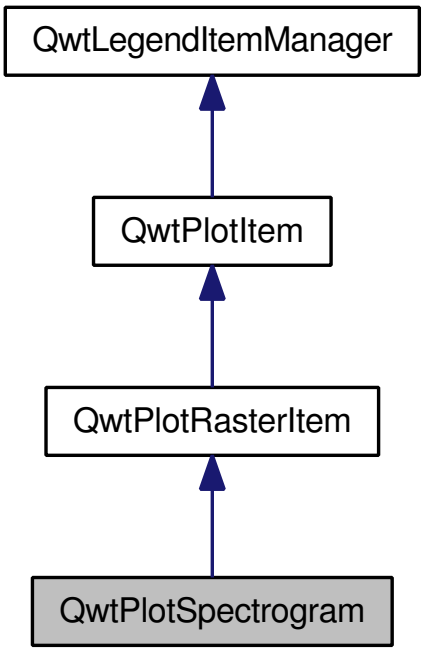
Reimplemented from QwtPlotItem.

12.64 QwtPlotSpectrogram Class Reference

A plot item, which displays a spectrogram.

```
#include <qwt_plot_spectrogram.h>
```

Inheritance diagram for QwtPlotSpectrogram:



Public Types

- enum DisplayMode {
  ImageMode = 1,
  ContourMode = 2 }

Public Member Functions

- QwtPlotSpectrogram (const QString &title=QString::null)
- virtual ~QwtPlotSpectrogram ()
- void setDisplayMode (DisplayMode, bool on=true)
- bool testDisplayMode (DisplayMode) const
- void setData (const QwtRasterData &data)
- const QwtRasterData & data () const
- void setColorMap (const QwtColorMap &)
- const QwtColorMap & colorMap () const
- virtual QwtDoubleRect boundingRect () const
- virtual QSize rasterHint (const QwtDoubleRect &) const
- void setDefaultContourPen (const QPen &)
- QPen defaultContourPen () const
- virtual QPen contourPen (double level) const
- void setConrecAttribute (QwtRasterData::ConrecAttribute, bool on)
- bool testConrecAttribute (QwtRasterData::ConrecAttribute) const
- void setContourLevels (const QwtValueList &)
- QwtValueList contourLevels () const
- virtual int rtti () const

- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &rect) const

Protected Member Functions

- virtual QImage renderImage (const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtDoubleRect &rect) const
- virtual QSize contourRasterSize (const QwtDoubleRect &, const QRect &) const
- virtual QwtRasterData::ContourLines renderContourLines (const QwtDoubleRect &rect, const QSize &raster) const
- virtual void drawContourLines (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QwtRasterData::ContourLines &lines) const

12.64.1 Detailed Description

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

In ContourMode contour lines are painted for the contour levels.

See also

QwtRasterData, QwtColorMap

12.64.2 Member Enumeration Documentation

12.64.2.1 enum QwtPlotSpectrogram::DisplayMode

The display mode controls how the raster data will be represented.

- ImageMode

  The values are mapped to colors using a color map.

- ContourMode

  The data is displayed using contour lines

When both modes are enabled the contour lines are painted on top of the spectrogram. The default setting enables ImageMode.

See also

setDisplayMode(), testDisplayMode()

12.64.3 Constructor & Destructor Documentation

12.64.3.1 QwtPlotSpectrogram::QwtPlotSpectrogram (const QString & *title* = `QString::null`) `[explicit]`

Sets the following item attributes:

- QwtPlotItem::AutoScale: true

- QwtPlotItem::Legend: false

The z value is initialized by 8.0.

Parameters

title Title

See also

QwtPlotItem::setItemAttribute(), QwtPlotItem::setZ()

12.64.3.2 QwtPlotSpectrogram::~QwtPlotSpectrogram () [virtual]

Destructor.

12.64.4 Member Function Documentation

12.64.4.1 QwtDoubleRect QwtPlotSpectrogram::boundingRect () const [virtual]

Returns

Bounding rect of the data

See also

QwtRasterData::boundingRect()

Reimplemented from QwtPlotItem.

12.64.4.2 const QwtColorMap & QwtPlotSpectrogram::colorMap () const

Returns

Color Map used for mapping the intensity values to colors

See also

setColorMap()

12.64.4.3 QwtValueList QwtPlotSpectrogram::contourLevels () const

Return the levels of the contour lines.

The levels are sorted in increasing order.

See also

contourLevels(), renderContourLines(), QwtRasterData::contourLines()

12.64.4.4 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()

12.64.4.5 QSize QwtPlotSpectrogram::contourRasterSize (const QwtDoubleRect & *area*, const QRect & *rect*) const [protected, virtual]

Return the raster to be used by the CONREC contour algorithm.

A larger size will improve the precisision of the CONREC algorithm, but will slow down the time that is needed to calculate the lines.

The default implementation returns rect.size() / 2 bounded to data().rasterHint().

Parameters

area Rect, where to calculate the contour lines

rect Rect in pixel coordinates, where to paint the contour lines

Returns

Raster to be used by the CONREC contour algorithm.

Note

The size will be bounded to rect.size().

See also

drawContourLines(), QwtRasterData::contourLines()

12.64.4.6 const QwtRasterData & QwtPlotSpectrogram::data () const

Returns

Spectrogram data

See also

setData()

12.64.4.7 QPen QwtPlotSpectrogram::defaultContourPen () const

Returns

Default contour pen

See also

setDefaultContourPen()

12.64.4.8 void QwtPlotSpectrogram::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the spectrogram.

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

See also

setDisplayMode(), renderImage(), QwtPlotRasterItem::draw(), drawContourLines()

Reimplemented from QwtPlotRasterItem.

12.64.4.9 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()

12.64.4.10 QSize QwtPlotSpectrogram::rasterHint (const QwtDoubleRect & *rect*) const [virtual]

Returns the recommended raster for a given rect.

F.e the raster hint is used to limit the resolution of the image that is rendered.

Parameters

rect Rect for the raster hint

Returns

data().rasterHint(rect)

Reimplemented from QwtPlotRasterItem.

12.64.4.11 QwtRasterData::ContourLines QwtPlotSpectrogram::renderContourLines (const QwtDoubleRect & *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

See also

contourLevels(), setConrecAttribute(), QwtRasterData::contourLines()

12.64.4.12 QImage QwtPlotSpectrogram::renderImage (const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QwtDoubleRect & *area*) const [protected, virtual]

Render an image from the data and color map.

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

Parameters

xMap X-Scale Map

yMap Y-Scale Map

area Area that should be rendered in scale coordinates.

Returns

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

See also

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

Implements QwtPlotRasterItem.

12.64.4.13 int QwtPlotSpectrogram::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotSpectrogram

Reimplemented from QwtPlotItem.

12.64.4.14 void QwtPlotSpectrogram::setColorMap (const QwtColorMap & *colorMap*)

Change the color map

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

Parameters

colorMap Color Map

See also

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

12.64.4.15 void QwtPlotSpectrogram::setConrecAttribute (QwtRasterData::ConrecAttribute *attribute*, bool *on*)

Modify an attribute of the CONREC algorithm, used to calculate the contour lines.

Parameters

attribute CONREC attribute

on On/Off

See also

testConrecAttribute(), renderContourLines(), QwtRasterData::contourLines()

12.64.4.16 void QwtPlotSpectrogram::setContourLevels (const QwtValueList & *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.

12.64.4.17 void QwtPlotSpectrogram::setData (const QwtRasterData & *data*)

Set the data to be displayed

Parameters

data Spectrogram Data

See also

data()

12.64.4.18 void QwtPlotSpectrogram::setDefaultContourPen (const QPen & *pen*)

Set the default pen for the contour lines.

If the spectrogram has a valid default contour pen a contour line is painted using the default contour pen. Otherwise (pen.style() == Qt::NoPen) the pen is calculated for each contour level using contourPen().

See also

defaultContourPen(), contourPen()

12.64.4.19 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()

12.64.4.20 bool QwtPlotSpectrogram::testConrecAttribute (QwtRasterData::ConrecAttribute *attribute*) const

Test an attribute of the CONREC algorithm, used to calculate the contour lines.

Parameters

attribute CONREC attribute

Returns

true, is enabled

See also

setConrecAttribute(), renderContourLines(), QwtRasterData::contourLines()

12.64.4.21 bool QwtPlotSpectrogram::testDisplayMode (DisplayMode *mode*) const

The display mode controls how the raster data will be represented.

Parameters

mode Display mode

Returns

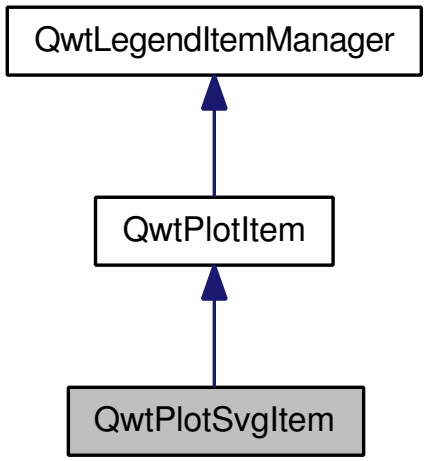
true if mode is enabled

12.65 QwtPlotSvgItem Class Reference

A plot item, which displays data in Scalable Vector Graphics (SVG) format.

`#include <qwt_plot_svgitem.h>`

Inheritance diagram for QwtPlotSvgItem:



Public Member Functions

- QwtPlotSvgItem (const QString &title=QString::null)
- QwtPlotSvgItem (const QwtText &title)
- virtual ~QwtPlotSvgItem ()
- bool loadFile (const QwtDoubleRect &, const QString &fileName)
- bool loadData (const QwtDoubleRect &, const QByteArray &)
- virtual QwtDoubleRect boundingRect () const
- virtual void draw (QPainter *p, const QwtScaleMap &xMap, const QwtScaleMap &yMap, const QRect &rect) const
- virtual int rtti () const

Protected Member Functions

- void render (QPainter *painter, const QwtDoubleRect &viewBox, const QRect &rect) const
- QwtDoubleRect viewBox (const QwtDoubleRect &area) const

12.65.1 Detailed Description

A plot item, which displays data in Scalable Vector Graphics (SVG) format. SVG images are often used to display maps

12.65.2 Constructor & Destructor Documentation

12.65.2.1 QwtPlotSvgItem::QwtPlotSvgItem (const QString & *title* = `QString::null`) `[explicit]`

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

title Title

12.65.2.2 QwtPlotSvgItem::QwtPlotSvgItem (const QwtText & *title*) `[explicit]`

Constructor.

Sets the following item attributes:

- QwtPlotItem::AutoScale: true
- QwtPlotItem::Legend: false

Parameters

title Title

12.65.2.3 QwtPlotSvgItem::~QwtPlotSvgItem () `[virtual]`

Destructor.

12.65.3 Member Function Documentation

12.65.3.1 QwtDoubleRect QwtPlotSvgItem::boundingRect () const `[virtual]`

Bounding rect of the item.

Reimplemented from QwtPlotItem.

12.65.3.2 void QwtPlotSvgItem::draw (QPainter * *painter*, const QwtScaleMap & *xMap*, const QwtScaleMap & *yMap*, const QRect & *canvasRect*) const [virtual]

Draw the SVG item

Parameters

painter Painter

xMap X-Scale Map

yMap Y-Scale Map

canvasRect Contents rect of the plot canvas

Implements QwtPlotItem.

12.65.3.3 bool QwtPlotSvgItem::loadData (const QwtDoubleRect & *rect*, const QByteArray & *data*)

Load SVG data

Parameters

rect Bounding rectangle

data in SVG format

Returns

true, if the SVG data could be loaded

12.65.3.4 bool QwtPlotSvgItem::loadFile (const QwtDoubleRect & *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

12.65.3.5 void QwtPlotSvgItem::render (QPainter * *painter*, const QwtDoubleRect & *viewBox*, const QRect & *rect*) const [protected]

Render the SVG data

Parameters

painter Painter

viewBox View Box, see QSvgRenderer::viewBox

rect Traget rectangle on the paint device

12.65.3.6 int QwtPlotSvgItem::rtti () const [virtual]

Returns

QwtPlotItem::Rtti_PlotSVG

Reimplemented from QwtPlotItem.

12.65.3.7 QwtDoubleRect QwtPlotSvgItem::viewBox (const QwtDoubleRect & *rect*) const [protected]

Calculate the viewBox from an rect and boundingRect().

Parameters

rect Rectangle in scale coordinates

Returns

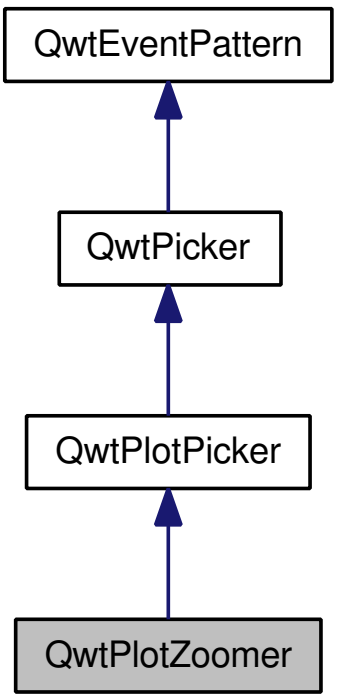
viewBox View Box, see QSvgRenderer::viewBox

12.66 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 x, double y)
- virtual void move (double x, double y)
- virtual void zoom (const QwtDoubleRect &)
- virtual void zoom (int up)

Signals

- void zoomed (const QwtDoubleRect &rect)

Public Member Functions

- QwtPlotZoomer (QwtPlotCanvas *, bool doReplot=true)
- QwtPlotZoomer (int xAxis, int yAxis, QwtPlotCanvas *, bool doReplot=true)
- QwtPlotZoomer (int xAxis, int yAxis, int selectionFlags, DisplayMode trackerMode, QwtPlotCanvas *, bool doReplot=true)
- virtual void setZoomBase (bool doReplot=true)
- virtual void setZoomBase (const QwtDoubleRect &)
- QwtDoubleRect zoomBase () const
- QwtDoubleRect zoomRect () const
- virtual void setAxis (int xAxis, int yAxis)
- void setMaxStackDepth (int)
- int maxStackDepth () const
- const QStack< QwtDoubleRect > & zoomStack () const
- void **setZoomStack** (const QStack< QwtDoubleRect > &, int zoomRectIndex=-1)
- uint zoomRectIndex () const
- virtual void setSelectionFlags (int)

Protected Member Functions

- virtual void rescale ()
- virtual QwtDoubleSize minZoomSize () const
- virtual void widgetMouseReleaseEvent (QMouseEvent *)
- virtual void widgetKeyPressEvent (QKeyEvent *)
- virtual void begin ()
- virtual bool end (bool ok=true)
- virtual bool accept (QwtPolygon &) const

12.66.1 Detailed Description

QwtPlotZoomer provides stacked zooming for a plot widget. QwtPlotZoomer offers rubberband selections on the plot canvas, translating the selected rectangles into plot coordinates and adjusting the axes to them. Zooming can repeated as often as possible, limited only by maxStackDepth() or minZoomSize(). Each rectangle is pushed on a stack.

Zoom rectangles can be selected depending on selectionFlags() using the mouse or keyboard (QwtEventPattern, QwtPickerMachine). QwtEventPattern::MouseSelect3/QwtEventPatternKeyUndo, or QwtEventPattern::MouseSelect6/QwtEventPatternKeyRedo walk up and down the zoom stack. QwtEventPattern::MouseSelect2 or QwtEventPattern::KeyHome unzoom to the initial size.

QwtPlotZoomer is tailored for plots with one x and y axis, but it is allowed to attach a second QwtPlotZoomer for the other axes.

Note

The realtime example includes an derived zoomer class that adds scrollbars to the plot canvas.

12.66.2 Constructor & Destructor Documentation

12.66.2.1 QwtPlotZoomer::QwtPlotZoomer (QwtPlotCanvas * *canvas*, bool *doReplot* = `true`) `[explicit]`

Create a zoomer for a plot canvas.

The zoomer is set to those x- and y-axis of the parent plot of the canvas that are enabled. If both or no x-axis are enabled, the picker is set to QwtPlot::xBottom. If both or no y-axis are enabled, it is set to QwtPlot::yLeft.

The selectionFlags() are set to QwtPicker::RectSelection | QwtPicker::ClickSelection, the tracker mode to QwtPicker::ActiveOnly.

Parameters

canvas Plot canvas to observe, also the parent object

doReplot Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

QwtPlot::autoReplot(), QwtPlot::replot(), setZoomBase()

12.66.2.2 QwtPlotZoomer::QwtPlotZoomer (int *xAxis*, int *yAxis*, QwtPlotCanvas * *canvas*, bool *doReplot* = `true`) `[explicit]`

Create a zoomer for a plot canvas.

The selectionFlags() are set to QwtPicker::RectSelection | QwtPicker::ClickSelection, the tracker mode to QwtPicker::ActiveOnly.

Parameters

xAxis X axis of the zoomer

yAxis Y axis of the zoomer

canvas Plot canvas to observe, also the parent object

doReplot Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

QwtPlot::autoReplot(), QwtPlot::replot(), setZoomBase()

12.66.2.3 QwtPlotZoomer::QwtPlotZoomer (int *xAxis*, int *yAxis*, int *selectionFlags*, DisplayMode *trackerMode*, QwtPlotCanvas * *canvas*, bool *doReplot* = `true`) `[explicit]`

Create a zoomer for a plot canvas.

Parameters

xAxis X axis of the zoomer

yAxis Y axis of the zoomer

selectionFlags Or'd value of QwtPicker::RectSelectionType and QwtPicker::SelectionMode. QwtPicker::RectSelection will be auto added.

trackerMode Tracker mode

canvas Plot canvas to observe, also the parent object

doReplot Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

QwtPicker, QwtPicker::setSelectionFlags(), QwtPicker::setRubberBand(), QwtPicker::setTrackerMode()
QwtPlot::autoReplot(), QwtPlot::replot(), setZoomBase()

12.66.3 Member Function Documentation

12.66.3.1 bool QwtPlotZoomer::accept (QwtPolygon & *pa*) const `[protected, virtual]`

Check and correct a selected rectangle.

Reject rectangles with a hight or width < 2, otherwise expand the selected rectangle to a minimum size of 11x11 and accept it.

Returns

true If rect is accepted, or has been changed to a accepted rectangle.

Reimplemented from QwtPicker.

12.66.3.2 void QwtPlotZoomer::begin () `[protected, virtual]`

Rejects selections, when the stack depth is too deep, or the zoomed rectangle is minZoomSize().

See also

minZoomSize(), maxStackDepth()

Reimplemented from QwtPicker.

12.66.3.3 bool QwtPlotZoomer::end (bool *ok* = `true`) `[protected, virtual]`

Expand the selected rectangle to minZoomSize() and zoom in if accepted.

See also

accept(), minZoomSize()

Reimplemented from QwtPlotPicker.

12.66.3.4 int QwtPlotZoomer::maxStackDepth () const

Returns

Maximal depth of the zoom stack.

See also

setMaxStackDepth()

12.66.3.5 QwtDoubleSize QwtPlotZoomer::minZoomSize () const [protected, virtual]

Limit zooming by a minimum rectangle.

Returns

zoomBase().width() / 10e4, zoomBase().height() / 10e4

12.66.3.6 void QwtPlotZoomer::move (double *x*, double *y*) [virtual, slot]

Move the the current zoom rectangle.

Parameters

x X value

y Y value

See also

QwtDoubleRect::move()

Note

The changed rectangle is limited by the zoom base

12.66.3.7 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

12.66.3.8 void QwtPlotZoomer::rescale () [protected, virtual]

Adjust the observed plot to zoomRect()

Note

Initiates QwtPlot::replot

12.66.3.9 void QwtPlotZoomer::setAxis (int *xAxis*, int *yAxis*) [virtual]

Reinitialize the axes, and set the zoom base to their scales.

Parameters

xAxis X axis

yAxis Y axis

Reimplemented from QwtPlotPicker.

12.66.3.10 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.

12.66.3.11 void QwtPlotZoomer::setSelectionFlags (int *flags*) [virtual]

Set the selection flags

Parameters

flags Or'd value of QwtPicker::RectSelectionType and QwtPicker::SelectionMode. The default value is QwtPicker::RectSelection & QwtPicker::ClickSelection.

See also

selectionFlags(), SelectionType, RectSelectionType, SelectionMode

Note

QwtPicker::RectSelection will be auto added.

Reimplemented from QwtPicker.

12.66.3.12 void QwtPlotZoomer::setZoomBase (const QwtDoubleRect & *base*) [virtual]

Set the initial size of the zoomer.

base is united with the current scaleRect() and the zoom stack is reinitalized with it as zoom base. plot is zoomed to scaleRect().

Parameters

base Zoom base

See also

zoomBase(), scaleRect()

12.66.3.13 void QwtPlotZoomer::setZoomBase (bool *doReplot* = true) [virtual]

Reinitialized the zoom stack with scaleRect() as base.

Parameters

doReplot Call replot for the attached plot before initializing the zoomer with its scales. This might be necessary, when the plot is in a state with pending scale changes.

See also

zoomBase(), scaleRect() QwtPlot::autoReplot(), QwtPlot::replot().

12.66.3.14 void QwtPlotZoomer::widgetKeyPressEvent (QKeyEvent * *ke*) [protected, virtual]

Qt::Key_Plus zooms in, Qt::Key_Minus zooms out one position on the zoom stack, Qt::Key_Escape zooms out to the zoom base.

Changes the current position on the stack, but doesn't pop any rectangle.

Note

The keys codes can be changed, using QwtEventPattern::setKeyPattern: 3, 4, 5

Reimplemented from QwtPicker.

12.66.3.15 void QwtPlotZoomer::widgetMouseReleaseEvent (QMouseEvent * *me*) [protected, virtual]

Qt::MidButton zooms out one position on the zoom stack, Qt::RightButton to the zoom base.

Changes the current position on the stack, but doesn't pop any rectangle.

Note

The mouse events can be changed, using QwtEventPattern::setMousePattern: 2, 1

Reimplemented from QwtPicker.

12.66.3.16 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 offest 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()

12.66.3.17 void QwtPlotZoomer::zoom (const QwtDoubleRect & *rect*) [virtual, slot]

Zoom in.

Clears all rectangles above the current position of the zoom stack and pushs the intersection of zoomRect() and the normalized rect on it.

Note

If the maximal stack depth is reached, zoom is ignored.
The zoomed signal is emitted.

12.66.3.18 QwtDoubleRect QwtPlotZoomer::zoomBase () const

Returns

Initial rectangle of the zoomer

See also

setZoomBase(), zoomRect()

12.66.3.19 void QwtPlotZoomer::zoomed (const QwtDoubleRect & *rect*) [signal]

A signal emitting the zoomRect(), when the plot has been zoomed in or out.

Parameters

rect Current zoom rectangle.

12.66.3.20 QwtDoubleRect QwtPlotZoomer::zoomRect () const

Rectangle at the current position on the zoom stack.

See also

zoomRectIndex(), scaleRect().

12.66.3.21 uint QwtPlotZoomer::zoomRectIndex () const

Returns

Index of current position of zoom stack.

12.66.3.22 const QwtZoomStack & QwtPlotZoomer::zoomStack () const

Return the zoom stack. zoomStack()[0] is the zoom base, zoomStack()[1] the first zoomed rectangle.

See also

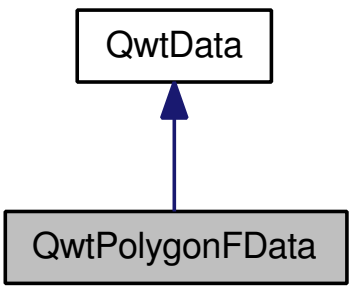
setZoomStack(), zoomRectIndex()

12.67 QwtPolygonFData Class Reference

Data class containing a single QwtArray<QwtDoublePoint> object.

`#include <qwt_data.h>`

Inheritance diagram for QwtPolygonFData:



Public Member Functions

- QwtPolygonFData (const QPolygonF &)
- QwtPolygonFData & operator= (const QwtPolygonFData &)
- virtual QwtData * copy () const
- virtual size_t size () const
- virtual double x (size_t i) const
- virtual double y (size_t i) const
- const QPolygonF & data () const

12.67.1 Detailed Description

Data class containing a single QwtArray<QwtDoublePoint> object.

12.67.2 Constructor & Destructor Documentation

12.67.2.1 QwtPolygonFData::QwtPolygonFData (const QPolygonF & *polygon*)

Constructor

Parameters

polygon Polygon data

See also

QwtPlotCurve::setData()

12.67.3 Member Function Documentation

12.67.3.1 QwtData * QwtPolygonFData::copy () const `[virtual]`

Returns

Pointer to a copy (virtual copy constructor)

Implements QwtData.

12.67.3.2 const QPolygonF & QwtPolygonFData::data () const

Returns

Point array

12.67.3.3 QwtPolygonFData & QwtPolygonFData::operator= (const QwtPolygonFData & *data*)

Assignment.

12.67.3.4 size_t QwtPolygonFData::size () const [virtual]

Returns

Size of the data set

Implements QwtData.

12.67.3.5 double QwtPolygonFData::x (size_t *i*) const [virtual]

Return the x value of data point i

Parameters

i Index

Returns

x X value of data point i

Implements QwtData.

12.67.3.6 double QwtPolygonFData::y (size_t *i*) const [virtual]

Return the y value of data point i

Parameters

i Index

Returns

y Y value of data point i

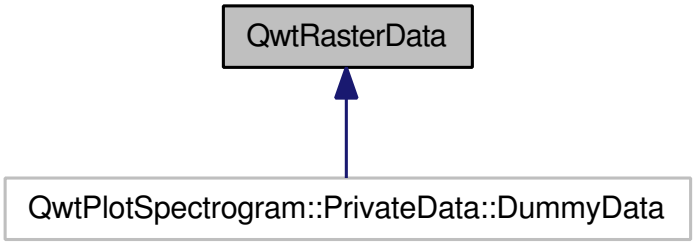
Implements QwtData.

12.68 QwtRasterData Class Reference

QwtRasterData defines an interface to any type of raster data.

`#include <qwt_raster_data.h>`

Inheritance diagram for QwtRasterData:



Public Types

- enum ConrecAttribute {
  IgnoreAllVerticesOnLevel = 1,
  IgnoreOutOfRange = 2 }
- typedef QMap< double, QPolygonF > **ContourLines**

Public Member Functions

- QwtRasterData ()
- QwtRasterData (const QwtDoubleRect &)
- virtual ~QwtRasterData ()
- virtual QwtRasterData * copy () const =0
- virtual void setBoundingRect (const QwtDoubleRect &)
- QwtDoubleRect boundingRect () const
- virtual QSize rasterHint (const QwtDoubleRect &) const
- virtual void initRaster (const QwtDoubleRect &, const QSize &raster)
- virtual void discardRaster ()
- virtual double value (double x, double y) const =0
- virtual QwtDoubleInterval range () const =0
- virtual ContourLines contourLines (const QwtDoubleRect &rect, const QSize &raster, const QList< double > &levels, int flags) const

12.68.1 Detailed Description

QwtRasterData defines an interface to any type of raster data. QwtRasterData is an abstract interface, that is used by QwtPlotRasterItem to find the values at the pixels of its raster.

Often a raster item is used to display values from a matrix. Then the derived raster data class needs to implement some sort of resampling, that maps the raster of the matrix into the requested raster of the raster item (depending on resolution and scales of the canvas).

12.68.2 Member Enumeration Documentation

12.68.2.1 enum QwtRasterData::ConrecAttribute

Attribute to modify the contour algorithm.

12.68.3 Constructor & Destructor Documentation

12.68.3.1 QwtRasterData::QwtRasterData ()

Constructor.

12.68.3.2 QwtRasterData::QwtRasterData (const QwtDoubleRect & *boundingRect*)

Constructor

Parameters

boundingRect Bounding rectangle

See also

setBoundingRect()

12.68.3.3 QwtRasterData::~QwtRasterData () `[virtual]`

Destructor.

12.68.4 Member Function Documentation

12.68.4.1 QwtDoubleRect QwtRasterData::boundingRect () const

Returns

Bounding rectangle

See also

boundingRect()

12.68.4.2 QwtRasterData::ContourLines QwtRasterData::contourLines (const QwtDoubleRect & *rect*, const QSize & *raster*, const QList< double > & *levels*, int *flags*) const `[virtual]`

Calculate contour lines

An adaption of CONREC, a simple contouring algorithm. http://local.wasp.uwa.edu.au/~pbourke/papers/c

12.68.4.3 virtual QwtRasterData* QwtRasterData::copy () const `[pure virtual]`

Clone the data.

12.68.4.4 void QwtRasterData::discardRaster () `[virtual]`

Discard a raster.

After the composition of an image QwtPlotSpectrogram calls discardRaster().

The default implementation does nothing, but if data has been loaded in initRaster(), it could deleted now.

See also

initRaster(), value()

12.68.4.5 void QwtRasterData::initRaster (const QwtDoubleRect &, const QSize & *raster*) [virtual]

Initialize a raster.

Before the composition of an image QwtPlotSpectrogram calls initRaster, announcing the area and its resolution that will be requested.

The default implementation does nothing, but for data sets that are stored in files, it might be good idea to reimplement initRaster, where the data is resampled and loaded into memory.

Parameters

rect Area of the raster

raster Number of horizontal and vertical pixels

See also

initRaster(), value()

12.68.4.6 virtual QwtDoubleInterval QwtRasterData::range () const [pure virtual]

Returns

the range of the values

12.68.4.7 QSize QwtRasterData::rasterHint (const QwtDoubleRect &) const [virtual]

Find the raster of the data for an area.

The resolution is the number of horizontal and vertical pixels that the data can return for an area. An invalid resolution indicates that the data can return values for any detail level.

The resolution will limit the size of the image that is rendered from the data. F.e. this might be important when printing a spectrogram to a A0 printer with 600 dpi.

The default implementation returns an invalid resolution (size)

Parameters

rect In most implementations the resolution of the data doesn't depend on the requested rectangle.

Returns

Resolution, as number of horizontal and vertical pixels

12.68.4.8 void QwtRasterData::setBoundingRect (const QwtDoubleRect & *boundingRect*) [virtual]

Set the bounding rect (== area, un plot coordinates)

Parameters

boundingRect Bounding rectangle

See also

boundingRect()

12.68.4.9 virtual double QwtRasterData::value (double *x*, double *y*) const [pure virtual]

Returns

the value at a raster position

Parameters

x X value in plot coordinates

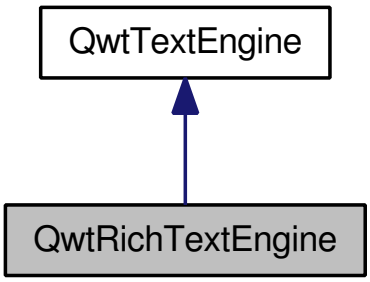
y Y value in plot coordinates

12.69 QwtRichTextEngine Class Reference

A text engine for Qt rich texts.

`#include <qwt_text_engine.h>`

Inheritance diagram for QwtRichTextEngine:



Public Member Functions

- QwtRichTextEngine ()
- virtual int heightForWidth (const QFont &font, int flags, const QString &text, int width) const
- virtual QSize textSize (const QFont &font, int flags, const QString &text) const
- virtual void draw (QPainter *painter, const QRect &rect, int flags, const QString &text) const
- virtual bool mightRender (const QString &) const
- virtual void textMargins (const QFont &, const QString &, int &left, int &right, int &top, int &bottom) const

12.69.1 Detailed Description

A text engine for Qt rich texts. QwtRichTextEngine renders Qt rich texts using the classes of the Scribe framework of Qt.

12.69.2 Constructor & Destructor Documentation

12.69.2.1 QwtRichTextEngine::QwtRichTextEngine ()

Constructor.

12.69.3 Member Function Documentation

12.69.3.1 void QwtRichTextEngine::draw (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) const [virtual]

Draw the text in a clipping rectangle

Parameters

painter Painter

rect Clipping rectangle

flags Bitwise OR of the flags like in for QPainter::drawText

text Text to be rendered

Implements QwtTextEngine.

12.69.3.2 int QwtRichTextEngine::heightForWidth (const QFont & *font*, int *flags*, const QString & *text*, int *width*) const [virtual]

Find the height for a given width

Parameters

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.

12.69.3.3 bool QwtRichTextEngine::mightRender (const QString & *text*) const [virtual]

Test if a string can be rendered by this text engine

Parameters

text Text to be tested

Returns

QStyleSheet::mightBeRichText(text);

Implements QwtTextEngine.

12.69.3.4 void QwtRichTextEngine::textMargins (const QFont &, const QString &, int & *left*, int & *right*, int & *top*, int & *bottom*) const [virtual]

Return margins around the texts

Parameters

left Return 0

right Return 0

top Return 0

bottom Return 0

Implements QwtTextEngine.

12.69.3.5 QSize QwtRichTextEngine::textSize (const QFont & *font*, int *flags*, const QString & *text*) const [virtual]

Returns the size, that is needed to render text

Parameters

font Font of the text

flags Bitwise OR of the flags used like in QPainter::drawText

text Text to be rendered

Returns

Caluclated size

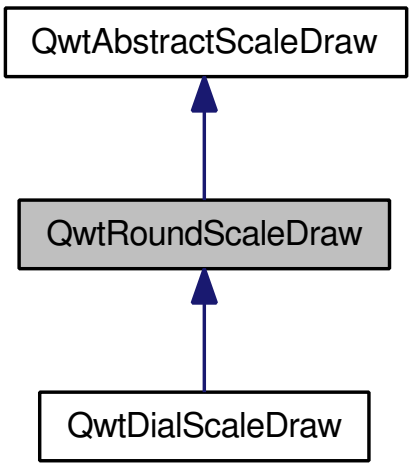
Implements QwtTextEngine.

12.70 QwtRoundScaleDraw Class Reference

A class for drawing round scales.

`#include <qwt_round_scale_draw.h>`

Inheritance diagram for QwtRoundScaleDraw:



Public Member Functions

- QwtRoundScaleDraw ()
- QwtRoundScaleDraw (const QwtRoundScaleDraw &)
- virtual ~QwtRoundScaleDraw ()
- QwtRoundScaleDraw & operator= (const QwtRoundScaleDraw &other)
- void setRadius (int radius)
- int radius () const
- void moveCenter (int x, int y)
- void moveCenter (const QPoint &)
- QPoint center () const
- void setAngleRange (double angle1, double angle2)
- virtual int extent (const QPen &, const QFont &) const

Protected Member Functions

- virtual void drawTick (QPainter *p, double val, int len) const
- virtual void drawBackbone (QPainter *p) const
- virtual void drawLabel (QPainter *p, double val) const

12.70.1 Detailed Description

A class for drawing round scales. QwtRoundScaleDraw can be used to draw round scales. The circle segment can be adjusted by QwtRoundScaleDraw::setAngleRange(). The geometry of the scale can be specified with QwtRoundScaleDraw::moveCenter() and QwtRoundScaleDraw::setRadius().

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

12.70.2 Constructor & Destructor Documentation

12.70.2.1 QwtRoundScaleDraw::QwtRoundScaleDraw ()

Constructor.

The range of the scale is initialized to [0, 100], The center is set to (50, 50) with a radius of 50. The angle range is set to [-135, 135].

12.70.2.2 QwtRoundScaleDraw::QwtRoundScaleDraw (const QwtRoundScaleDraw & *other*)

Copy constructor.

12.70.2.3 QwtRoundScaleDraw::~QwtRoundScaleDraw () `[virtual]`

Destructor.

12.70.3 Member Function Documentation

12.70.3.1 QPoint QwtRoundScaleDraw::center () const

Get the center of the scale.

12.70.3.2 void QwtRoundScaleDraw::drawBackbone (QPainter * *painter*) const `[protected, virtual]`

Draws the baseline of the scale

Parameters

painter Painter

See also

drawTick(), drawLabel()

Implements QwtAbstractScaleDraw.

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

Draws the label for a major scale tick

Parameters

painter Painter

value Value

See also

drawTick(), drawBackbone()

Implements QwtAbstractScaleDraw.

12.70.3.4 void QwtRoundScaleDraw::drawTick (QPainter * *painter*, double *value*, int *len*) const [protected, virtual]

Draw a tick

Parameters

painter Painter

value Value of the tick

len Lenght of the tick

See also

drawBackbone(), drawLabel()

Implements QwtAbstractScaleDraw.

12.70.3.5 int QwtRoundScaleDraw::extent (const QPen & *pen*, const QFont & *font*) const [virtual]

Calculate the extent of the scale

The extent is the distcance 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

pen Pen that is used for painting backbone and ticks

font Font used for painting the labels

See also

setMinimumExtent(), minimumExtent()

Warning

The implemented algo is not too smart and calculates only an upper limit, that might be a few pixels too large

Implements QwtAbstractScaleDraw.

12.70.3.6 void QwtRoundScaleDraw::moveCenter (const QPoint & *center*)

Move the center of the scale draw, leaving the radius unchanged

Parameters

center New center

See also

setRadius()

12.70.3.7 void QwtRoundScaleDraw::moveCenter (int *x*, int *y*) [inline]

Move the center of the scale draw, leaving the radius unchanged.

12.70.3.8 QwtRoundScaleDraw & QwtRoundScaleDraw::operator= (const QwtRoundScaleDraw & *other*)

Assignment operator.

12.70.3.9 int QwtRoundScaleDraw::radius () const

Get the radius

Radius is the radius of the backbone without ticks and labels.

See also

setRadius(), extent()

12.70.3.10 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 than 359 degrees above or below min(angle1, angle2), scale marks will not be drawn.
- If you need a counterclockwise scale, use QwtScaleDiv::setRange

12.70.3.11 void QwtRoundScaleDraw::setRadius (int *radius*)

Change of radius the scale

Radius is the radius of the backbone without ticks and labels.

Parameters

radius New Radius

See also

moveCenter()

12.71 QwtScaleArithmetic Class Reference

Arithmetic including a tolerance.

`#include <qwt_scale_engine.h>`

Static Public Member Functions

- static int compareEps (double value1, double value2, double intervalSize)
- static double ceilEps (double value, double intervalSize)
- static double floorEps (double value, double intervalSize)
- static double divideEps (double interval, double steps)
- static double ceil125 (double x)
- static double floor125 (double x)

12.71.1 Detailed Description

Arithmetic including a tolerance.

12.71.2 Member Function Documentation

12.71.2.1 double QwtScaleArithmetic::ceil125 (double *x*) [static]

Find the smallest value out of {1,2,5}∗10^n with an integer number n which is greater than or equal to x

Parameters

x Input value

12.71.2.2 double QwtScaleArithmetic::ceilEps (double *value*, double *intervalSize*) [static]

Ceil a value, relative to an interval

Parameters

value Value to ceil

intervalSize Interval size

See also

floorEps()

12.71.2.3 int QwtScaleArithmetic::compareEps (double *value1*, double *value2*, double *intervalSize*) [static]

Compare 2 values, relative to an interval.

Values are "equal", when : $\cdot value2 - value1 <= abs(intervalSize * 10e^{-6})$

Parameters

value1 First value to compare

value2 Second value to compare

intervalSize interval size

Returns

0: if equal, -1: if value2 > value1, 1: if value1 > value2

12.71.2.4 double QwtScaleArithmetic::divideEps (double *intervalSize*, double *numSteps*) [static]

Divide an interval into steps.

$stepSize = (intervalSize - intervalSize * 10e^{-6})/numSteps$

Parameters

intervalSize Interval size

numSteps Number of steps

Returns

Step size

12.71.2.5 double QwtScaleArithmetic::floor125 (double *x*) [static]

Find the largest value out of {1,2,5}*10^n with an integer number n which is smaller than or equal to x.

Parameters

x Input value

12.71.2.6 double QwtScaleArithmetic::floorEps (double *value*, double *intervalSize*) [static]

Floor a value, relative to an interval

Parameters

value Value to floor

intervalSize Interval size

See also

floorEps()

12.72 QwtScaleDiv Class Reference

A class representing a scale division.

`#include <qwt_scale_div.h>`

Public Types

- enum TickType {
  NoTick = -1,
  MinorTick,
  MediumTick,
  MajorTick,
  NTickTypes }

Public Member Functions

- QwtScaleDiv ()
- QwtScaleDiv (const QwtDoubleInterval &, QwtValueList[NTickTypes])
- QwtScaleDiv (double lowerBound, double upperBound, QwtValueList[NTickTypes])
- int operator== (const QwtScaleDiv &s) const
- int operator!= (const QwtScaleDiv &s) const
- void setInterval (double lowerBound, double upperBound)
- void setInterval (const QwtDoubleInterval &)
- QwtDoubleInterval interval () const
- double lowerBound () const
- double upperBound () const
- double range () const
- bool contains (double v) const
- void setTicks (int type, const QwtValueList &)
- const QwtValueList & ticks (int type) const
- void invalidate ()
- bool isValid () const
- void invert ()

12.72.1 Detailed Description

A class representing a scale division. A scale division consists of its limits and 3 list of tick values qualified as major, medium and minor ticks.

In most cases scale divisions are calculated by a QwtScaleEngine.

See also

subDivideInto(), subDivide()

12.72.2 Member Enumeration Documentation

12.72.2.1 enum QwtScaleDiv::TickType

Scale tick types.

12.72.3 Constructor & Destructor Documentation

12.72.3.1 QwtScaleDiv::QwtScaleDiv () `[explicit]`

Construct an invalid QwtScaleDiv instance.

12.72.3.2 QwtScaleDiv::QwtScaleDiv (const QwtDoubleInterval & *interval*, QwtValueList *ticks*[NTickTypes]) `[explicit]`

Construct QwtScaleDiv instance.

Parameters

interval Interval

ticks List of major, medium and minor ticks

12.72.3.3 QwtScaleDiv::QwtScaleDiv (double *lowerBound*, double *upperBound*, QwtValueList *ticks*[NTickTypes]) `[explicit]`

Construct QwtScaleDiv instance.

Parameters

lowerBound First interval limit

upperBound Second interval limit

ticks List of major, medium and minor ticks

12.72.4 Member Function Documentation

12.72.4.1 bool QwtScaleDiv::contains (double *value*) const

Return if a value is between lowerBound() and upperBound()

Parameters

value Value

Returns

true/false

12.72.4.2 QwtDoubleInterval QwtScaleDiv::interval () const [inline]

Returns

lowerBound -> upperBound

12.72.4.3 void QwtScaleDiv::invalidate ()

Invalidate the scale division.

12.72.4.4 void QwtScaleDiv::invert ()

Invert the scale divison.

12.72.4.5 bool QwtScaleDiv::isValid () const

Check if the scale division is valid.

12.72.4.6 double QwtScaleDiv::lowerBound () const [inline]

Returns

lower bound

See also

upperBound()

12.72.4.7 int QwtScaleDiv::operator!= (const QwtScaleDiv & *s*) const

Inequality.

Returns

true if this instance is not equal to s

12.72.4.8 int QwtScaleDiv::operator== (const QwtScaleDiv & *other*) const

Equality operator.

Returns

true if this instance is equal to other

12.72.4.9 double QwtScaleDiv::range () const [inline]

Returns

upperBound() - lowerBound()

12.72.4.10 void QwtScaleDiv::setInterval (const QwtDoubleInterval & *interval*)

Change the interval

Parameters

interval Interval

12.72.4.11 void QwtScaleDiv::setInterval (double *lowerBound*, double *upperBound*) [inline]

Change the interval

Parameters

lowerBound lower bound
upperBound upper bound

12.72.4.12 void QwtScaleDiv::setTicks (int *type*, const QwtValueList & *ticks*)

Assign ticks

Parameters

type MinorTick, MediumTick or MajorTick
ticks Values of the tick positions

12.72.4.13 const QwtValueList & QwtScaleDiv::ticks (int *type*) const

Return a list of ticks

Parameters

type MinorTick, MediumTick or MajorTick

12.72.4.14 double QwtScaleDiv::upperBound () const [inline]

Returns

upper bound

See also

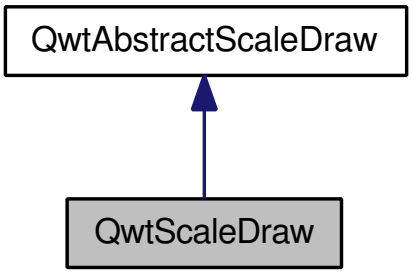
lowerBound()

12.73 QwtScaleDraw Class Reference

A class for drawing scales.

`#include <qwt_scale_draw.h>`

Inheritance diagram for QwtScaleDraw:



Public Types

- enum Alignment {
  BottomScale,
  TopScale,
  LeftScale,
  RightScale }

Public Member Functions

- QwtScaleDraw ()
- QwtScaleDraw (const QwtScaleDraw &)

- virtual ~QwtScaleDraw ()
- QwtScaleDraw & operator= (const QwtScaleDraw &other)
- void getBorderDistHint (const QFont &, int &start, int &end) const
- int minLabelDist (const QFont &) const
- int minLength (const QPen &, const QFont &) const
- virtual int extent (const QPen &, const QFont &) const
- void move (int x, int y)
- void move (const QPoint &)
- void setLength (int length)
- Alignment alignment () const
- void setAlignment (Alignment)
- Qt::Orientation orientation () const
- QPoint pos () const
- int length () const
- void setLabelAlignment (Qt::Alignment)
- Qt::Alignment labelAlignment () const
- void setLabelRotation (double rotation)
- double labelRotation () const
- int maxLabelHeight (const QFont &) const
- int maxLabelWidth (const QFont &) const
- QPoint labelPosition (double val) const
- QRect labelRect (const QFont &, double val) const
- QSize labelSize (const QFont &, double val) const
- QRect boundingLabelRect (const QFont &, double val) const

Protected Member Functions

- QMatrix labelMatrix (const QPoint &, const QSize &) const
- virtual void drawTick (QPainter *p, double val, int len) const
- virtual void drawBackbone (QPainter *p) const
- virtual void drawLabel (QPainter *p, double val) const

12.73.1 Detailed Description

A class for drawing scales. QwtScaleDraw can be used to draw linear or logarithmic scales. A scale has a position, an alignment and a length, which can be specified . The labels can be rotated and aligned to the ticks using setLabelRotation() and setLabelAlignment().

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

12.73.2 Member Enumeration Documentation

12.73.2.1 enum QwtScaleDraw::Alignment

Alignment of the scale draw

See also

setAlignment(), alignment()

12.73.3 Constructor & Destructor Documentation

12.73.3.1 QwtScaleDraw::QwtScaleDraw ()

Constructor.

The range of the scale is initialized to [0, 100], The position is at (0, 0) with a length of 100. The orientation is QwtAbstractScaleDraw::Bottom.

12.73.3.2 QwtScaleDraw::QwtScaleDraw (const QwtScaleDraw & *other*)

Copy constructor.

12.73.3.3 QwtScaleDraw::~QwtScaleDraw () `[virtual]`

Destructor.

12.73.4 Member Function Documentation

12.73.4.1 QwtScaleDraw::Alignment QwtScaleDraw::alignment () const

Return alignment of the scale

See also

setAlignment()

12.73.4.2 QRect QwtScaleDraw::boundingLabelRect (const QFont & *font*, double *value*) const

Find the bounding rect for the label. The coordinates of the rect are absolute coordinates (calculated from pos()). in direction of the tick.

Parameters

font Font used for painting

value Value

See also

labelRect()

12.73.4.3 void QwtScaleDraw::drawBackbone (QPainter * *painter*) const `[protected, virtual]`

Draws the baseline of the scale

Parameters

painter Painter

See also

drawTick(), drawLabel()

Implements QwtAbstractScaleDraw.

12.73.4.4 void QwtScaleDraw::drawLabel (QPainter * *painter*, double *value*) const [protected, virtual]

Draws the label for a major scale tick

Parameters

painter Painter

value Value

See also

drawTick(), drawBackbone(), boundingLabelRect()

Implements QwtAbstractScaleDraw.

12.73.4.5 void QwtScaleDraw::drawTick (QPainter * *painter*, double *value*, int *len*) const [protected, virtual]

Draw a tick

Parameters

painter Painter

value Value of the tick

len Lenght of the tick

See also

drawBackbone(), drawLabel()

Implements QwtAbstractScaleDraw.

12.73.4.6 int QwtScaleDraw::extent (const QPen & *pen*, const QFont & *font*) const [virtual]

Calculate the width/height that is needed for a vertical/horizontal scale.

The extent is calculated from the pen width of the backbone, the major tick length, the spacing and the maximum width/height of the labels.

Parameters

pen Pen that is used for painting backbone and ticks

font Font used for painting the labels

See also

minLength()

Implements QwtAbstractScaleDraw.

12.73.4.7 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

12.73.4.8 Qt::Alignment QwtScaleDraw::labelAlignment () const

Returns

the label flags

See also

setLabelAlignment(), labelRotation()

12.73.4.9 QMatrix QwtScaleDraw::labelMatrix (const QPoint & *pos*, const QSize & *size*) const [protected]

Calculate the matrix that is needed to paint a label depending on its alignment and rotation.

Parameters

pos Position where to paint the label

size Size of the label

See also

setLabelAlignment(), setLabelRotation()

12.73.4.10 QPoint QwtScaleDraw::labelPosition (double *value*) const

Find the position, where to paint a label

The position has a distance of majTickLength() + spacing() + 1 from the backbone. The direction depends on the alignment()

Parameters

value Value

12.73.4.11 QRect QwtScaleDraw::labelRect (const QFont & *font*, double *value*) const

Find the bounding rect for the label. The coordinates of the rect are relative to spacing + ticklength from the backbone in direction of the tick.

Parameters

font Font used for painting

value Value

12.73.4.12 double QwtScaleDraw::labelRotation () const

Returns

the label rotation

See also

setLabelRotation(), labelAlignment()

12.73.4.13 QSize QwtScaleDraw::labelSize (const QFont & *font*, double *value*) const

Calculate the size that is needed to draw a label

Parameters

font Label font

value Value

12.73.4.14 int QwtScaleDraw::length () const

Returns

the length of the backbone

See also

setLength(), pos()

12.73.4.15 int QwtScaleDraw::maxLabelHeight (const QFont & *font*) const

Parameters

font Font

Returns

the maximum height of a label

12.73.4.16 int QwtScaleDraw::maxLabelWidth (const QFont & *font*) const

Parameters

font Font

Returns

the maximum width of a label

12.73.4.17 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()

12.73.4.18 int QwtScaleDraw::minLength (const QPen & *pen*, const QFont & *font*) const

Calculate the minimum length that is needed to draw the scale

Parameters

pen Pen that is used for painting backbone and ticks

font Font used for painting the labels

See also

extent()

12.73.4.19 void QwtScaleDraw::move (const QPoint & *pos*)

Move the position of the scale.

The meaning of the parameter pos depends on the alignment:

QwtScaleDraw::LeftScale The origin is the topmost point of the backbone. The backbone is a vertical line. Scale marks and labels are drawn at the left of the backbone.

QwtScaleDraw::RightScale The origin is the topmost point of the backbone. The backbone is a vertical line. Scale marks and labels are drawn at the right of the backbone.

QwtScaleDraw::TopScale The origin is the leftmost point of the backbone. The backbone is a horizontal line. Scale marks and labels are drawn above the backbone.

QwtScaleDraw::BottomScale The origin is the leftmost point of the backbone. The backbone is a horizontal line Scale marks and labels are drawn below the backbone.

Parameters

pos Origin of the scale

See also

pos(), setLength()

12.73.4.20 void QwtScaleDraw::move (int *x*, int *y*) [inline]

Move the position of the scale

See also

move(const QPoint &)

12.73.4.21 QwtScaleDraw & QwtScaleDraw::operator= (const QwtScaleDraw & *other*)

Assignment operator.

12.73.4.22 Qt::Orientation QwtScaleDraw::orientation () const

Return the orientation

TopScale, BottomScale are horizontal (Qt::Horizontal) scales, LeftScale, RightScale are vertical (Qt::Vertical) scales.

See also

alignment()

12.73.4.23 QPoint QwtScaleDraw::pos () const

Returns

Origin of the scale

See also

move(), length()

12.73.4.24 void QwtScaleDraw::setAlignment (Alignment *align*)

Set the alignment of the scale

The default alignment is QwtScaleDraw::BottomScale

See also

alignment()

12.73.4.25 void QwtScaleDraw::setLabelAlignment (Qt::Alignment *alignment*)

Change the label flags.

Labels are aligned to the point ticklength + spacing away from the backbone.

The alignment is relative to the orientation of the label text. In case of an flags of 0 the label will be aligned depending on the orientation of the scale:

QwtScaleDraw::TopScale: Qt::AlignHCenter | Qt::AlignTop

QwtScaleDraw::BottomScale: Qt::AlignHCenter | Qt::AlignBottom

QwtScaleDraw::LeftScale: Qt::AlignLeft | Qt::AlignVCenter

QwtScaleDraw::RightScale: Qt::AlignRight | Qt::AlignVCenter

Changing the alignment is often necessary for rotated labels.

Parameters

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().

12.73.4.26 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().

12.73.4.27 void QwtScaleDraw::setLength (int *length*)

Set the length of the backbone.

The length doesn't include the space needed for overlapping labels.

See also

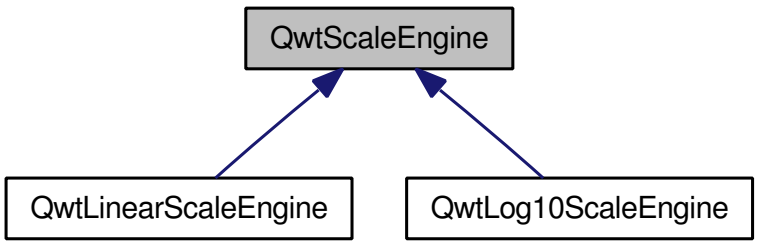
move(), minLabelDist()

12.74 QwtScaleEngine Class Reference

Base class for scale engines.

`#include <qwt_scale_engine.h>`

Inheritance diagram for QwtScaleEngine:



Public Types

- enum Attribute {
  NoAttribute = 0,
  IncludeReference = 1,
  Symmetric = 2,
  Floating = 4,
  Inverted = 8 }

Public Member Functions

- QwtScaleEngine ()
- virtual ~QwtScaleEngine ()
- void setAttribute (Attribute, bool on=true)
- bool testAttribute (Attribute) const
- void setAttributes (int)
- int attributes () const
- void setReference (double reference)
- double reference () const
- void setMargins (double lower, double upper)
- 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 maxMajSteps, int maxMinSteps, double stepSize=0.0) const =0
- virtual QwtScaleTransformation * transformation () const =0

Protected Member Functions

- bool contains (const QwtDoubleInterval &, double val) const
- QwtValueList strip (const QwtValueList &, const QwtDoubleInterval &) const
- double divideInterval (double interval, int numSteps) const
- QwtDoubleInterval buildInterval (double v) const

12.74.1 Detailed Description

Base class for scale engines. A scale engine trys to find "reasonable" ranges and step sizes for scales.

The layout of the scale can be varied with setAttribute().

Qwt offers implementations for logarithmic (log10) and linear scales. Contributions for other types of scale engines (date/time, log2 ...) are welcome.

12.74.2 Member Enumeration Documentation

12.74.2.1 enum QwtScaleEngine::Attribute

- IncludeReference

  Build a scale which includes the reference() value.

- Symmetric

  Build a scale which is symmetric to the reference() value.

- Floating

  The endpoints of the scale are supposed to be equal the outmost included values plus the specified margins (see setMargins()). If this attribute is *not* set, the endpoints of the scale will be integer multiples of the step size.

- Inverted

  Turn the scale upside down.

See also

setAttribute(), testAttribute(), reference(), lowerMargin(), upperMargin()

12.74.3 Constructor & Destructor Documentation

12.74.3.1 QwtScaleEngine::QwtScaleEngine () [explicit]

Constructor.

12.74.3.2 QwtScaleEngine::~QwtScaleEngine () [virtual]

Destructor.

12.74.4 Member Function Documentation

12.74.4.1 int QwtScaleEngine::attributes () const

Return the scale attributes

See also

Attribute, setAttributes(), testAttribute()

12.74.4.2 virtual void QwtScaleEngine::autoScale (int *maxNumSteps*, double & *x1*, double & *x2*, double & *stepSize*) const [pure virtual]

Align and divide an interval

Parameters

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 QwtLinearScaleEngine, and QwtLog10ScaleEngine.

12.74.4.3 QwtDoubleInterval QwtScaleEngine::buildInterval (double *v*) const [protected]

Build an interval for a value.

In case of v == 0.0 the interval is [-0.5, 0.5], otherwide it is [0.5 ∗ v, 1.5 ∗ v]

12.74.4.4 bool QwtScaleEngine::contains (const QwtDoubleInterval & *interval*, double *value*) const [protected]

Check if an interval "contains" a value

Parameters

interval Interval

value Value

See also

QwtScaleArithmetic::compareEps()

12.74.4.5 double QwtScaleEngine::divideInterval (double *intervalSize*, int *numSteps*) const [protected]

Calculate a step size for an interval size

Parameters

intervalSize Interval size

numSteps Number of steps

Returns

Step size

12.74.4.6 virtual QwtScaleDiv QwtScaleEngine::divideScale (double *x1*, double *x2*, int *maxMajSteps*, int *maxMinSteps*, double *stepSize* = 0.0) const [pure virtual]

Calculate a scale division.

Parameters

x1 First interval limit

x2 Second interval limit

maxMajSteps Maximum for the number of major steps

maxMinSteps Maximum number of minor steps

stepSize Step size. If stepSize == 0.0, the scaleEngine calculates one.

Implemented in QwtLinearScaleEngine, and QwtLog10ScaleEngine.

12.74.4.7 double QwtScaleEngine::lowerMargin () const

Returns

the margin at the lower end of the scale The default margin is 0.

See also

setMargins()

12.74.4.8 double QwtScaleEngine::reference () const

Returns

the reference value

See also

setReference(), setAttribute()

12.74.4.9 void QwtScaleEngine::setAttribute (Attribute *attribute*, bool *on* = `true`)

Change a scale attribute

Parameters

attribute Attribute to change

on On/Off

See also

Attribute, testAttribute()

12.74.4.10 void QwtScaleEngine::setAttributes (int *attributes*)

Change the scale attribute

Parameters

attributes Set scale attributes

See also

Attribute, attributes()

12.74.4.11 void QwtScaleEngine::setMargins (double *lower*, double *upper*)

Specify margins at the scale's endpoints.

Parameters

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

- QwtLog10ScaleEngine measures the margins in decades.

See also

upperMargin(), lowerMargin()

12.74.4.12 void QwtScaleEngine::setReference (double *r*)

Specify a reference point.

Parameters

r new reference value

The reference point is needed if options IncludeReference or Symmetric are active. Its default value is 0.0.

See also

Attribute

12.74.4.13 QwtValueList QwtScaleEngine::strip (const QwtValueList & *ticks*, const QwtDoubleInterval & *interval*) const [protected]

Remove ticks from a list, that are not inside an interval

Parameters

ticks Tick list

interval Interval

Returns

Stripped tick list

12.74.4.14 bool QwtScaleEngine::testAttribute (Attribute *attribute*) const

Check if a attribute is set.

Parameters

attribute Attribute to be tested

See also

Attribute, setAttribute()

12.74.4.15 virtual QwtScaleTransformation* QwtScaleEngine::transformation () const [pure virtual]

Returns

a transformation

Implemented in QwtLinearScaleEngine, and QwtLog10ScaleEngine.

12.74.4.16 double QwtScaleEngine::upperMargin () const

Returns

the margin at the upper end of the scale The default margin is 0.

See also

setMargins()

12.75 QwtScaleMap Class Reference

A scale map.

`#include <qwt_scale_map.h>`

Public Member Functions

- QwtScaleMap ()
- QwtScaleMap (const QwtScaleMap &)
- ~QwtScaleMap ()
- QwtScaleMap & operator= (const QwtScaleMap &)
- void setTransformation (QwtScaleTransformation *)
- const QwtScaleTransformation * transformation () const
- void setPaintInterval (int p1, int p2)
- void setPaintXInterval (double p1, double p2)
- void setScaleInterval (double s1, double s2)
- int transform (double x) const
- double invTransform (double i) const
- double xTransform (double x) const
- double p1 () const
- double p2 () const
- double s1 () const
- double s2 () const
- double pDist () const
- double sDist () const

Public Attributes

- QT_STATIC_CONST double **LogMin** = 1.0e-150
- QT_STATIC_CONST double **LogMax** = 1.0e150

12.75.1 Detailed Description

A scale map. QwtScaleMap offers transformations from a scale into a paint interval and vice versa.

12.75.2 Constructor & Destructor Documentation

12.75.2.1 QwtScaleMap::QwtScaleMap ()

Constructor.

The scale and paint device intervals are both set to [0,1].

12.75.2.2 QwtScaleMap::QwtScaleMap (const QwtScaleMap & *other*)

Copy constructor.

12.75.2.3 QwtScaleMap::~QwtScaleMap ()

Destructor

12.75.3 Member Function Documentation

12.75.3.1 double QwtScaleMap::invTransform (double *p*) const [inline]

Transform an paint device value into a value in the interval of the scale.

Parameters

p Value relative to the coordinates of the paint device

See also

transform()

12.75.3.2 QwtScaleMap & QwtScaleMap::operator= (const QwtScaleMap & *other*)

Assignment operator.

12.75.3.3 double QwtScaleMap::p1 () const [inline]

Returns

First border of the paint interval

12.75.3.4 double QwtScaleMap::p2 () const [inline]

Returns

Second border of the paint interval

12.75.3.5 double QwtScaleMap::pDist () const [inline]

Returns

qwtAbs(p2() - p1())

12.75.3.6 double QwtScaleMap::s1 () const [inline]

Returns

First border of the scale interval

12.75.3.7 double QwtScaleMap::s2 () const [inline]

Returns

Second border of the scale interval

12.75.3.8 double QwtScaleMap::sDist () const [inline]

Returns

qwtAbs(s2() - s1())

12.75.3.9 void QwtScaleMap::setPaintInterval (int *p1*, int *p2*)

Specify the borders of the paint device interval.

Parameters

p1 first border

p2 second border

12.75.3.10 void QwtScaleMap::setPaintXInterval (double *p1*, double *p2*)

Specify the borders of the paint device interval.

Parameters

p1 first border

p2 second border

12.75.3.11 void QwtScaleMap::setScaleInterval (double *s1*, double *s2*)

Specify the borders of the scale interval.

Parameters

s1 first border

s2 second border

Warning

logarithmic scales might be aligned to [LogMin, LogMax]

12.75.3.12 void QwtScaleMap::setTransformation (QwtScaleTransformation * *transformation*)

Initialize the map with a transformation

12.75.3.13 int QwtScaleMap::transform (double *s*) const [inline]

Transform a point related to the scale interval into an point related to the interval of the paint device and round it to an integer. (In Qt <= 3.x paint devices are integer based.)

Parameters

s Value relative to the coordinates of the scale

See also

xTransform()

12.75.3.14 const QwtScaleTransformation * QwtScaleMap::transformation () const

Get the transformation.

12.75.3.15 double QwtScaleMap::xTransform (double *s*) const [inline]

Transform a point related to the scale interval into an point related to the interval of the paint device

Parameters

s Value relative to the coordinates of the scale

12.76 QwtScaleTransformation Class Reference

Operations for linear or logarithmic (base 10) transformations.

`#include <qwt_scale_map.h>`

Public Types

- enum **Type** {
  Linear,
  Log10,
  Other }

Public Member Functions

- QwtScaleTransformation (Type type)
- virtual ~QwtScaleTransformation ()
- virtual double xForm (double x, double s1, double s2, double p1, double p2) const
- virtual double invXForm (double x, double s1, double s2, double p1, double p2) const
- Type type () const
- virtual QwtScaleTransformation * copy () const

12.76.1 Detailed Description

Operations for linear or logarithmic (base 10) transformations.

12.76.2 Constructor & Destructor Documentation

12.76.2.1 QwtScaleTransformation::QwtScaleTransformation (Type *type*)

Constructor for a linear transformation.

12.76.2.2 QwtScaleTransformation::~QwtScaleTransformation () [virtual]

Destructor.

12.76.3 Member Function Documentation

12.76.3.1 QwtScaleTransformation ∗ QwtScaleTransformation::copy () const [virtual]

Create a clone of the transformation.

12.76.3.2 double QwtScaleTransformation::invXForm (double *p*, double *p1*, double *p2*, double *s1*, double *s2*) const [virtual]

Transform a value from a linear to a logarithmic interval.

Parameters

p value related to the linear interval [p1, p2]

p1 first border of linear interval

p2 second border of linear interval

s1 first border of logarithmic interval

s2 second border of logarithmic interval

Returns

exp((p - p1) / (p2 - p1) ∗ log(s2 / s1)) ∗ s1;

12.76.3.3 QwtScaleTransformation::Type QwtScaleTransformation::type () const [inline]

Returns

Transformation type

12.76.3.4 double QwtScaleTransformation::xForm (double *s*, double *s1*, double *s2*, double *p1*, double *p2*) const [virtual]

Transform a value between 2 linear intervals.

Parameters

s value related to the interval [s1, s2]

s1 first border of scale interval

s2 second border of scale interval

p1 first border of target interval

p2 second border of target interval

Returns

linear mapping: p1 + (p2 - p1) / (s2 - s1) ∗ (s - s1)

log10 mapping: p1 + (p2 - p1) / log(s2 / s1) ∗ log(s / s1)

12.77 QwtScaleWidget Class Reference

A Widget which contains a scale.

`#include <qwt_scale_widget.h>`

Signals

- void scaleDivChanged ()

Public Member Functions

- QwtScaleWidget (QWidget *parent=NULL)
- QwtScaleWidget (QwtScaleDraw::Alignment, QWidget *parent=NULL)
- virtual ~QwtScaleWidget ()
- void setTitle (const QString &title)
- void setTitle (const QwtText &title)
- QwtText title () const
- void setBorderDist (int start, int end)
- int startBorderDist () const
- int endBorderDist () const
- void getBorderDistHint (int &start, int &end) const
- void getMinBorderDist (int &start, int &end) const
- void setMinBorderDist (int start, int end)
- void setMargin (int)
- int margin () const
- void setSpacing (int td)
- int spacing () const
- void setPenWidth (int)
- int penWidth () const
- void setScaleDiv (QwtScaleTransformation *, const QwtScaleDiv &sd)
- void setScaleDraw (QwtScaleDraw *)
- const QwtScaleDraw * scaleDraw () const
- QwtScaleDraw * scaleDraw ()
- void setLabelAlignment (Qt::Alignment)
- void setLabelRotation (double rotation)
- void **setColorBarEnabled** (bool)
- bool **isColorBarEnabled** () const
- void **setColorBarWidth** (int)
- int **colorBarWidth** () const
- void **setColorMap** (const QwtDoubleInterval &, const QwtColorMap &)
- QwtDoubleInterval **colorBarInterval** () const
- const QwtColorMap & **colorMap** () const
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- int titleHeightForWidth (int width) const
- int dimForLength (int length, const QFont &scaleFont) const
- void **drawColorBar** (QPainter *painter, const QRect &rect) const
- void drawTitle (QPainter *painter, QwtScaleDraw::Alignment, const QRect &rect) const
- void setAlignment (QwtScaleDraw::Alignment)
- QwtScaleDraw::Alignment alignment () const
- QRect **colorBarRect** (const QRect &) const

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗e)
- virtual void resizeEvent (QResizeEvent ∗e)
- void draw (QPainter ∗p) const
- void scaleChange ()
- void layoutScale (bool update=true)

12.77.1 Detailed Description

A Widget which contains a scale. This Widget can be used to decorate composite widgets with a scale.

12.77.2 Constructor & Destructor Documentation

12.77.2.1 QwtScaleWidget::QwtScaleWidget (QWidget ∗ *parent* = NULL) [explicit]

Create a scale with the position QwtScaleWidget::Left.

Parameters

parent Parent widget

12.77.2.2 QwtScaleWidget::QwtScaleWidget (QwtScaleDraw::Alignment *align*, QWidget ∗ *parent* = NULL) [explicit]

Constructor.

Parameters

align Alignment.

parent Parent widget

12.77.2.3 QwtScaleWidget::∼QwtScaleWidget () [virtual]

Destructor.

12.77.3 Member Function Documentation

12.77.3.1 QwtScaleDraw::Alignment QwtScaleWidget::alignment () const

Returns

position

See also

setPosition()

12.77.3.2 int QwtScaleWidget::dimForLength (int *length*, const QFont & *scaleFont*) const

Find the minimum dimension for a given length. dim is the height, length the width seen in direction of the title.

Parameters

length width for horizontal, height for vertical scales

scaleFont Font of the scale

Returns

height for horizontal, width for vertical scales

12.77.3.3 void QwtScaleWidget::draw (QPainter * *p*) const [protected]

draw the scale

12.77.3.4 void QwtScaleWidget::drawTitle (QPainter * *painter*, QwtScaleDraw::Alignment *align*, const QRect & *rect*) const

Rotate and paint a title according to its position into a given rectangle.

Parameters

painter Painter

align Alignment

rect Bounding rectangle

12.77.3.5 int QwtScaleWidget::endBorderDist () const

Returns

end border distance

See also

setBorderDist()

12.77.3.6 void QwtScaleWidget::getBorderDistHint (int & *start*, int & *end*) const

Calculate a hint for the border distances.

This member function calculates the distance of the scale's endpoints from the widget borders which is required for the mark labels to fit into the widget. The maximum of this distance an the minimum border distance is returned.

Warning

- The minimum border distance depends on the font.

See also

setMinBorderDist(), getMinBorderDist(), setBorderDist()

12.77.3.7 void QwtScaleWidget::getMinBorderDist (int & *start*, int & *end*) const

Get the minimum value for the distances of the scale's endpoints from the widget borders.

See also

setMinBorderDist(), getBorderDistHint()

12.77.3.8 void QwtScaleWidget::layoutScale (bool *update* = `true`) `[protected]`

Recalculate the scale's geometry and layout based on.

12.77.3.9 int QwtScaleWidget::margin () const

Returns

margin

See also

setMargin()

12.77.3.10 QSize QwtScaleWidget::minimumSizeHint () const `[virtual]`

Returns

a minimum size hint

12.77.3.11 void QwtScaleWidget::paintEvent (QPaintEvent * *e*) [protected, virtual]

paintEvent

12.77.3.12 int QwtScaleWidget::penWidth () const

Returns

Scale pen width

See also

setPenWidth()

12.77.3.13 void QwtScaleWidget::resizeEvent (QResizeEvent * *e*) [protected, virtual]

resizeEvent

12.77.3.14 void QwtScaleWidget::scaleChange () [protected]

Notify a change of the scale.

This virtual function can be overloaded by derived classes. The default implementation updates the geometry and repaints the widget.

12.77.3.15 void QwtScaleWidget::scaleDivChanged () [signal]

Signal emitted, whenever the scale divison changes.

12.77.3.16 QwtScaleDraw * QwtScaleWidget::scaleDraw ()

scaleDraw of this scale

See also

QwtScaleDraw::setScaleDraw()

12.77.3.17 const QwtScaleDraw * QwtScaleWidget::scaleDraw () const

scaleDraw of this scale

See also

setScaleDraw(), QwtScaleDraw::setScaleDraw()

12.77.3.18 void QwtScaleWidget::setAlignment (QwtScaleDraw::Alignment *alignment*)

Change the alignment

Parameters

alignment New alignment

See also

alignment()

12.77.3.19 void QwtScaleWidget::setBorderDist (int *dist1*, int *dist2*)

Specify distances of the scale's endpoints from the widget's borders. The actual borders will never be less than minimum border distance.

Parameters

dist1 Left or top Distance

dist2 Right or bottom distance

See also

borderDist()

12.77.3.20 void QwtScaleWidget::setLabelAlignment (Qt::Alignment *alignment*)

Change the alignment for the labels.

See also

QwtScaleDraw::setLabelAlignment(), setLabelRotation()

12.77.3.21 void QwtScaleWidget::setLabelRotation (double *rotation*)

Change the rotation for the labels. See QwtScaleDraw::setLabelRotation().

Parameters

rotation Rotation

See also

QwtScaleDraw::setLabelRotation(), setLabelFlags()

12.77.3.22 void QwtScaleWidget::setMargin (int *margin*)

Specify the margin to the colorBar/base line.

Parameters

margin Margin

See also

margin()

12.77.3.23 void QwtScaleWidget::setMinBorderDist (int *start*, int *end*)

Set a minimum value for the distances of the scale's endpoints from the widget borders. This is useful to avoid that the scales are "jumping", when the tick labels or their positions change often.

Parameters

start Minimum for the start border

end Minimum for the end border

See also

getMinBorderDist(), getBorderDistHint()

12.77.3.24 void QwtScaleWidget::setPenWidth (int *width*)

Specify the width of the scale pen.

Parameters

width Pen width

See also

penWidth()

12.77.3.25 void QwtScaleWidget::setScaleDiv (QwtScaleTransformation * *transformation*, const QwtScaleDiv & *scaleDiv*)

Assign a scale division.

The scale division determines where to set the tick marks.

Parameters

transformation Transformation, needed to translate between scale and pixal values

scaleDiv Scale Division

See also

For more information about scale divisions, see QwtScaleDiv.

12.77.3.26 void QwtScaleWidget::setScaleDraw (QwtScaleDraw * *sd*)

Set a scale draw sd has to be created with new and will be deleted in ~QwtScaleWidget() or the next call of setScaleDraw().

Parameters

sd ScaleDraw object

See also

scaleDraw()

12.77.3.27 void QwtScaleWidget::setSpacing (int *spacing*)

Specify the distance between color bar, scale and title.

Parameters

spacing Spacing

See also

spacing()

12.77.3.28 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.

12.77.3.29 void QwtScaleWidget::setTitle (const QString & *title*)

Give title new text contents

Parameters

title New title

See also

title(), setTitle(const QwtText &);

12.77.3.30 QSize QwtScaleWidget::sizeHint () const [virtual]

Returns

a size hint

12.77.3.31 int QwtScaleWidget::spacing () const

Returns

distance between scale and title

See also

setMargin()

12.77.3.32 int QwtScaleWidget::startBorderDist () const

Returns

start border distance

See also

setBorderDist()

12.77.3.33 QwtText QwtScaleWidget::title () const

Returns

title

See also

setTitle()

12.77.3.34 int QwtScaleWidget::titleHeightForWidth (int *width*) const

Find the height of the title for a given width.

Parameters

width Width

Returns

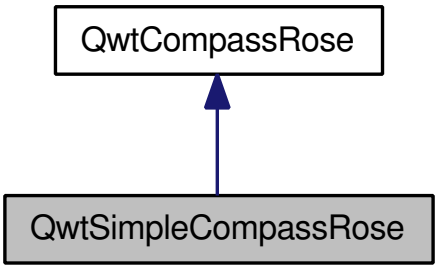
height Height

12.78 QwtSimpleCompassRose Class Reference

A simple rose for QwtCompass.

`#include <qwt_compass_rose.h>`

Inheritance diagram for QwtSimpleCompassRose:



Public Member Functions

- QwtSimpleCompassRose (int numThorns=8, int numThornLevels=-1)
- void setWidth (double w)
- double width () const
- void setNumThorns (int count)
- int numThorns () const
- void setNumThornLevels (int count)
- int numThornLevels () const
- void **setShrinkFactor** (double factor)
- double **shrinkFactor** () const
- virtual void draw (QPainter *, const QPoint &center, int radius, double north, QPalette::ColorGroup=QPalette::Active) const

Static Public Member Functions

- static void drawRose (QPainter *, const QPalette &, const QPoint &center, int radius, double origin, double width, int numThorns, int numThornLevels, double shrinkFactor)

12.78.1 Detailed Description

A simple rose for QwtCompass.

12.78.2 Constructor & Destructor Documentation

12.78.2.1 QwtSimpleCompassRose::QwtSimpleCompassRose (int *numThorns* = 8, int *numThornLevels* = -1)

Constructor

Parameters

numThorns Number of thorns

numThornLevels Number of thorn levels

12.78.3 Member Function Documentation

12.78.3.1 void QwtSimpleCompassRose::draw (QPainter * *painter*, const QPoint & *center*, int *radius*, double *north*, QPalette::ColorGroup *cg* = QPalette::Active) const [virtual]

Draw the rose

Parameters

painter Painter

center Center point

radius Radius of the rose

north Position

cg Color group

Implements QwtCompassRose.

12.78.3.2 void QwtSimpleCompassRose::drawRose (QPainter * *painter*, const QPalette & *palette*, const QPoint & *center*, int *radius*, double *north*, double *width*, int *numThorns*, int *numThornLevels*, double *shrinkFactor*) [static]

Draw the rose

Parameters

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

12.78.3.3 int QwtSimpleCompassRose::numThornLevels () const

Returns

Number of thorn levels

See also

setNumThorns(), setNumThornLevels()

12.78.3.4 int QwtSimpleCompassRose::numThorns () const

Returns

Number of thorns

See also

setNumThorns(), setNumThornLevels()

12.78.3.5 void QwtSimpleCompassRose::setNumThornLevels (int *numThornLevels*)

Set the of thorns levels

Parameters

numThornLevels Number of thorns levels

See also

setNumThorns(), numThornLevels()

12.78.3.6 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()

12.78.3.7 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

12.78.3.8 double QwtSimpleCompassRose::width () const [inline]

See also

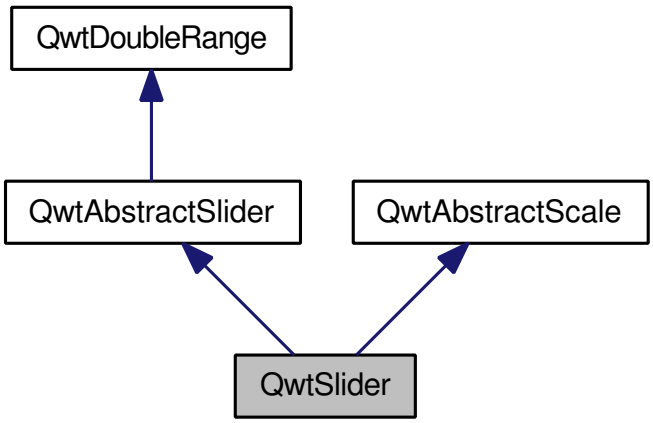
setWidth()

12.79 QwtSlider Class Reference

The Slider Widget.

`#include <qwt_slider.h>`

Inheritance diagram for QwtSlider:



Public Types

- enum ScalePos {
  NoScale,
  LeftScale,
  RightScale,
  TopScale,
  BottomScale }
- enum BGSTYLE {
  BgTrough = 0x1,
  BgSlot = 0x2,
  BgBoth = BgTrough | BgSlot }

Public Member Functions

- QwtSlider (QWidget *parent, Qt::Orientation=Qt::Horizontal, ScalePos=NoScale, BGSTYLE bgStyle=BgTrough)
- virtual void setOrientation (Qt::Orientation)
- void setBgStyle (BGSTYLE)

- BGSTYLE bgStyle () const
- void setScalePosition (ScalePos s)
- ScalePos scalePosition () const
- int thumbLength () const
- int thumbWidth () const
- int borderWidth () const
- void setThumbLength (int l)
- void setThumbWidth (int w)
- void setBorderWidth (int bw)
- void setMargins (int x, int y)
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- void setScaleDraw (QwtScaleDraw *)
- const QwtScaleDraw * scaleDraw () const

Protected Member Functions

- virtual double getValue (const QPoint &p)
- virtual void getScrollMode (const QPoint &p, int &scrollMode, int &direction)
- void draw (QPainter *p, const QRect &update_rect)
- virtual void drawSlider (QPainter *p, const QRect &r)
- virtual void drawThumb (QPainter *p, const QRect &, int pos)
- virtual void resizeEvent (QResizeEvent *e)
- virtual void paintEvent (QPaintEvent *e)
- virtual void valueChange ()
- virtual void rangeChange ()
- virtual void scaleChange ()
- virtual void fontChange (const QFont &oldFont)
- void layoutSlider (bool update=true)
- int xyPosition (double v) const
- QwtScaleDraw * scaleDraw ()

12.79.1 Detailed Description

The Slider Widget. QwtSlider is a slider widget which operates on an interval of type double. QwtSlider supports different layouts as well as a scale.

See also

QwtAbstractSlider and QwtAbstractScale for the descriptions of the inherited members.

12.79.2 Member Enumeration Documentation

12.79.2.1 enum QwtSlider::BGSTYLE

Background style.

See also

QwtSlider()

12.79.2.2 enum QwtSlider::ScalePos

Scale position. QwtSlider tries to enforce valid combinations of its orientation and scale position:

- Qt::Horizonal combines with NoScale, TopScale and BottomScale
- Qt::Vertical combines with NoScale, LeftScale and RightScale

See also

QwtSlider()

12.79.3 Constructor & Destructor Documentation

12.79.3.1 QwtSlider::QwtSlider (QWidget * *parent*, Qt::Orientation *orientation* = `Qt::Horizontal`, ScalePos *scalePos* = `NoScale`, BGSTYLE *bgStyle* = `BgTrough`) `[explicit]`

Constructor.

Parameters

parent parent widget

orientation Orientation of the slider. Can be Qt::Horizontal or Qt::Vertical. Defaults to Qt::Horizontal.

scalePos Position of the scale. Defaults to QwtSlider::NoScale.

bgStyle Background style. QwtSlider::BgTrough draws the slider button in a trough, QwtSlider::BgSlot draws a slot underneath the button. An or-combination of both may also be used. The default is QwtSlider::BgTrough.

QwtSlider enforces valid combinations of its orientation and scale position. If the combination is invalid, the scale position will be set to NoScale. Valid combinations are:

- Qt::Horizonal with NoScale, TopScale, or BottomScale;
- Qt::Vertical with NoScale, LeftScale, or RightScale.

12.79.4 Member Function Documentation

12.79.4.1 QwtSlider::BGSTYLE QwtSlider::bgStyle () const

Returns

the background style.

12.79.4.2 int QwtSlider::borderWidth () const

Returns

the border width.

12.79.4.3 void QwtSlider::draw (QPainter * *p*, const QRect & *update_rect*) [protected]

Draw the QwtSlider.

12.79.4.4 void QwtSlider::drawSlider (QPainter * *painter*, const QRect & *r*) [protected, virtual]

Draw the slider into the specified rectangle.

Parameters

painter Painter

r Rectangle

12.79.4.5 void QwtSlider::drawThumb (QPainter * *painter*, const QRect & *sliderRect*, int *pos*) [protected, virtual]

Draw the thumb at a position

Parameters

painter Painter

sliderRect Bounding rectangle of the slider

pos Position of the slider thumb

12.79.4.6 void QwtSlider::fontChange (const QFont & *oldFont*) [protected, virtual]

Notify change in font.

12.79.4.7 void QwtSlider::getScrollMode (const QPoint & *p*, int & *scrollMode*, int & *direction*) [protected, virtual]

Determine scrolling mode and direction.

Parameters

p point

scrollMode Scrolling mode

direction Direction

Implements QwtAbstractSlider.

12.79.4.8 double QwtSlider::getValue (const QPoint & *pos*) [protected, virtual]

Determine the value corresponding to a specified mouse location.

Parameters

pos Mouse position

Implements QwtAbstractSlider.

12.79.4.9 void QwtSlider::layoutSlider (bool *update_geometry* = true) [protected]

Recalculate the slider's geometry and layout based on the current rect and fonts.

Parameters

update_geometry notify the layout system and call update to redraw the scale

12.79.4.10 QSize QwtSlider::minimumSizeHint () const [virtual]

Return a minimum size hint.

Warning

The return value of QwtSlider::minimumSizeHint() depends on the font and the scale.

12.79.4.11 void QwtSlider::paintEvent (QPaintEvent * *event*) [protected, virtual]

Qt paint event

Parameters

event Paint event

12.79.4.12 void QwtSlider::rangeChange () [protected, virtual]

Notify change of range.

Reimplemented from QwtDoubleRange.

12.79.4.13 void QwtSlider::resizeEvent (QResizeEvent * *e*) [protected, virtual]

Qt resize event.

12.79.4.14 void QwtSlider::scaleChange () [protected, virtual]

Notify changed scale.

Reimplemented from QwtAbstractScale.

12.79.4.15 QwtScaleDraw * QwtSlider::scaleDraw () [protected]

Returns

the scale draw of the slider

See also

setScaleDraw()

12.79.4.16 const QwtScaleDraw * QwtSlider::scaleDraw () const

Returns

the scale draw of the slider

See also

setScaleDraw()

12.79.4.17 QwtSlider::ScalePos QwtSlider::scalePosition () const

Return the scale position.

12.79.4.18 void QwtSlider::setBgStyle (BGSTYLE *st*)

Set the background style.

12.79.4.19 void QwtSlider::setBorderWidth (int *bd*)

Change the slider's border width.

Parameters

bd border width

12.79.4.20 void QwtSlider::setMargins (int *xMargin*, int *yMargin*)

Set distances between the widget's border and internals.

Parameters

xMargin Horizontal margin

yMargin Vertical margin

12.79.4.21 void QwtSlider::setOrientation (Qt::Orientation *o*) [virtual]

Set the orientation.

Parameters

o Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.

If the new orientation and the old scale position are an invalid combination, the scale position will be set to QwtSlider::NoScale.

See also

QwtAbstractSlider::orientation()

Reimplemented from QwtAbstractSlider.

12.79.4.22 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().

12.79.4.23 void QwtSlider::setScalePosition (ScalePos *s*)

Change the scale position (and slider orientation).

Parameters

s Position of the scale.

A valid combination of scale position and orientation is enforced:

- if the new scale position is Left or Right, the scale orientation will become Qt::Vertical;
- if the new scale position is Bottom or Top the scale orientation will become Qt::Horizontal;
- if the new scale position is QwtSlider::NoScale, the scale orientation will not change.

12.79.4.24 void QwtSlider::setThumbLength (int *thumbLength*)

Set the slider's thumb length.

Parameters

thumbLength new length

12.79.4.25 void QwtSlider::setThumbWidth (int *w*)

Change the width of the thumb.

Parameters

w new width

12.79.4.26 QSize QwtSlider::sizeHint () const [virtual]

Returns

QwtSlider::minimumSizeHint()

12.79.4.27 int QwtSlider::thumbLength () const

Returns

the thumb length.

12.79.4.28 int QwtSlider::thumbWidth () const

Returns

the thumb width.

12.79.4.29 void QwtSlider::valueChange () [protected, virtual]

Notify change of value.

Reimplemented from QwtAbstractSlider.

12.79.4.30 int QwtSlider::xyPosition (double *value*) const [protected]

Find the x/y position for a given value v

Parameters

value Value

12.80 QwtSpline Class Reference

A class for spline interpolation.

`#include <qwt_spline.h>`

Public Types

- enum SplineType {
  Natural,
  Periodic }

Public Member Functions

- QwtSpline ()
- QwtSpline (const QwtSpline &)
- ~QwtSpline ()
- QwtSpline & operator= (const QwtSpline &)
- void setSplineType (SplineType)
- SplineType splineType () const
- bool setPoints (const QPolygonF &points)
- QPolygonF points () const
- void reset ()
- bool isValid () const
- double value (double x) const
- const QwtArray< double > & coefficientsA () const
- const QwtArray< double > & coefficientsB () const
- const QwtArray< double > & coefficientsC () const

Protected Member Functions

- bool buildNaturalSpline (const QPolygonF &)
- bool buildPeriodicSpline (const QPolygonF &)

Protected Attributes

- PrivateData * **d_data**

12.80.1 Detailed Description

A class for spline interpolation. The QwtSpline class is used for cubical spline interpolation. Two types of splines, natural and periodic, are supported.

Usage:

1. First call setPoints() to determine the spline coefficients for a tabulated function y(x).
2. After the coefficients have been set up, the interpolated function value for an argument x can be determined by calling QwtSpline::value().

Example:

```
#include <qwt_spline.h>

QPolygonF interpolate(const QPolygonF& points, int numValues)
{
    QwtSpline spline;
    if ( !spline.setPoints(points) )
        return points;

    QPolygonF interpolatedPoints(numValues);

    const double delta =
        (points[numPoints - 1].x() - points[0].x()) / (points.size() - 1);
    for(i = 0; i < points.size(); i++)  / interpolate
    {
        const double x = points[0].x() + i * delta;
        interpolatedPoints[i].setX(x);
        interpolatedPoints[i].setY(spline.value(x));
    }
    return interpolatedPoints;
}
```

12.80.2 Member Enumeration Documentation

12.80.2.1 enum QwtSpline::SplineType

Spline type.

12.80.3 Constructor & Destructor Documentation

12.80.3.1 QwtSpline::QwtSpline ()

Constructor.

12.80.3.2 QwtSpline::QwtSpline (const QwtSpline & *other*)

Copy constructor

Parameters

other Spline used for initilization

12.80.3.3 QwtSpline::~QwtSpline ()

Destructor.

12.80.4 Member Function Documentation

12.80.4.1 bool QwtSpline::buildNaturalSpline (const QPolygonF & *points*) [protected]

Determines the coefficients for a natural spline.

Returns

true if successful

12.80.4.2 bool QwtSpline::buildPeriodicSpline (const QPolygonF & *points*) [protected]

Determines the coefficients for a periodic spline.

Returns

true if successful

12.80.4.3 const QwtArray< double > & QwtSpline::coefficientsA () const

Returns

A coefficients

12.80.4.4 const QwtArray< double > & QwtSpline::coefficientsB () const

Returns

B coefficients

12.80.4.5 const QwtArray< double > & QwtSpline::coefficientsC () const

Returns

C coefficients

12.80.4.6 bool QwtSpline::isValid () const

True if valid.

12.80.4.7 QwtSpline & QwtSpline::operator= (const QwtSpline & *other*)

Assignment operator

Parameters

other Spline used for initilization

12.80.4.8 QPolygonF QwtSpline::points () const

Return points passed by setPoints

12.80.4.9 void QwtSpline::reset ()

Free allocated memory and set size to 0.

12.80.4.10 bool QwtSpline::setPoints (const QPolygonF & *points*)

Calculate the spline coefficients.

Depending on the value of *periodic*, this function will determine the coefficients for a natural or a periodic spline and store them internally.

Parameters

points Points

Returns

true if successful

Warning

The sequence of x (but not y) values has to be strictly monotone increasing, which means `points[i].x() < points[i+1].x()`. If this is not the case, the function will return false

12.80.4.11 void QwtSpline::setSplineType (SplineType *splineType*)

Select the algorithm used for calculating the spline

Parameters

splineType Spline type

See also

splineType()

12.80.4.12 QwtSpline::SplineType QwtSpline::splineType () const

Returns

the spline type

See also

setSplineType()

12.80.4.13 double QwtSpline::value (double *x*) const

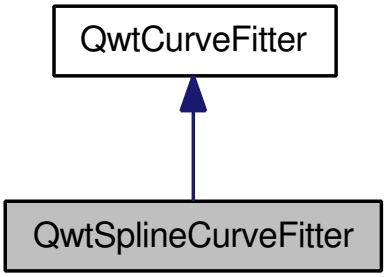
Calculate the interpolated function value corresponding to a given argument x.

12.81 QwtSplineCurveFitter Class Reference

A curve fitter using cubic splines.

`#include <qwt_curve_fitter.h>`

Inheritance diagram for QwtSplineCurveFitter:



Public Types

- enum **FitMode** {
  Auto,
  Spline,
  ParametricSpline }

Public Member Functions

- QwtSplineCurveFitter ()
- virtual ~QwtSplineCurveFitter ()
- void setFitMode (FitMode)
- FitMode fitMode () const
- void **setSpline** (const QwtSpline &)
- const QwtSpline & **spline** () const
- QwtSpline & **spline** ()
- void setSplineSize (int size)
- int splineSize () const
- virtual QPolygonF fitCurve (const QPolygonF &) const

12.81.1 Detailed Description

A curve fitter using cubic splines.

12.81.2 Constructor & Destructor Documentation

12.81.2.1 QwtSplineCurveFitter::QwtSplineCurveFitter ()

Constructor.

12.81.2.2 QwtSplineCurveFitter::~QwtSplineCurveFitter () `[virtual]`

Destructor.

12.81.3 Member Function Documentation

12.81.3.1 QPolygonF QwtSplineCurveFitter::fitCurve (const QPolygonF & *points*) const `[virtual]`

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

Parameters

points Series of data points

Returns

Curve points

Implements QwtCurveFitter.

12.81.3.2 QwtSplineCurveFitter::FitMode QwtSplineCurveFitter::fitMode () const

Returns

Mode representing a spline algorithm

See also

setFitMode()

12.81.3.3 void QwtSplineCurveFitter::setFitMode (FitMode *mode*)

Select the algorithm used for building the spline

Parameters

mode Mode representing a spline algorithm

See also

fitMode()

12.81.3.4 void QwtSplineCurveFitter::setSplineSize (int *splineSize*)

Assign a spline size (has to be at least 10 points)

Parameters

splineSize Spline size

See also

splineSize()

12.81.3.5 int QwtSplineCurveFitter::splineSize () const

Returns

Spline size

See also

setSplineSize()

12.82 QwtSymbol Class Reference

A class for drawing symbols.

`#include <qwt_symbol.h>`

Public Types

- enum Style {
  NoSymbol = -1,
  Ellipse,
  Rect,
  Diamond,
  Triangle,
  DTriangle,
  UTriangle,
  LTriangle,
  RTriangle,
  Cross,
  XCross,
  HLine,
  VLine,
  Star1,
  Star2,
  Hexagon,
  StyleCnt }

Public Member Functions

- QwtSymbol ()
- QwtSymbol (Style st, const QBrush &bd, const QPen &pn, const QSize &s)
- virtual ~QwtSymbol ()
- bool operator!= (const QwtSymbol &) const
- virtual bool operator== (const QwtSymbol &) const
- virtual QwtSymbol * clone () const
- void setSize (const QSize &s)
- void setSize (int a, int b=-1)
- void setBrush (const QBrush &b)
- void setPen (const QPen &p)
- void setStyle (Style s)
- const QBrush & brush () const
- const QPen & pen () const
- const QSize & size () const
- Style style () const
- void draw (QPainter *p, const QPoint &pt) const
- void draw (QPainter *p, int x, int y) const
- virtual void draw (QPainter *p, const QRect &r) const

12.82.1 Detailed Description

A class for drawing symbols.

12.82.2 Member Enumeration Documentation

12.82.2.1 enum QwtSymbol::Style

Style

See also

setStyle(), style()

12.82.3 Constructor & Destructor Documentation

12.82.3.1 QwtSymbol::QwtSymbol ()

Default Constructor

The symbol is constructed with gray interior, black outline with zero width, no size and style 'NoSymbol'.

12.82.3.2 QwtSymbol::QwtSymbol (QwtSymbol::Style *style*, const QBrush & *brush*, const QPen & *pen*, const QSize & *size*)

Constructor.

Parameters

style Symbol Style

brush brush to fill the interior

pen outline pen

size size

12.82.3.3 QwtSymbol::~QwtSymbol () `[virtual]`

Destructor.

12.82.4 Member Function Documentation

12.82.4.1 const QBrush& QwtSymbol::brush () const `[inline]`

Return Brush.

12.82.4.2 QwtSymbol * QwtSymbol::clone () const `[virtual]`

Allocate and return a symbol with the same attributes

Returns

Cloned symbol

12.82.4.3 void QwtSymbol::draw (QPainter * *painter*, const QRect & *r*) const [virtual]

Draw the symbol into a bounding rectangle.

This function assumes that the painter has been initialized with brush and pen before. This allows a much more performant implementation when painting many symbols with the same brush and pen like in curves.

Parameters

painter Painter

r Bounding rectangle

12.82.4.4 void QwtSymbol::draw (QPainter * *p*, int *x*, int *y*) const

Draw the symbol at a point (x,y).

12.82.4.5 void QwtSymbol::draw (QPainter * *painter*, const QPoint & *pos*) const

Draw the symbol at a specified point.

Parameters

painter Painter

pos Center of the symbol

12.82.4.6 bool QwtSymbol::operator!= (const QwtSymbol & *other*) const

!= operator

12.82.4.7 bool QwtSymbol::operator== (const QwtSymbol & *other*) const [virtual]

== operator

12.82.4.8 const QPen& QwtSymbol::pen () const [inline]

Return Pen.

12.82.4.9 void QwtSymbol::setBrush (const QBrush & *brush*)

Assign a brush.

The brush is used to draw the interior of the symbol.

Parameters

brush Brush

12.82.4.10 void QwtSymbol::setPen (const QPen & *pen*)

Assign a pen

The pen is used to draw the symbol's outline.

The width of non cosmetic pens is scaled according to the resolution of the paint device.

Parameters

pen Pen

See also

pen(), setBrush(), QwtPainter::scaledPen()

12.82.4.11 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)

12.82.4.12 void QwtSymbol::setSize (const QSize & *size*)

Set the symbol's size

Parameters

size Size

12.82.4.13 void QwtSymbol::setStyle (QwtSymbol::Style *s*)

Specify the symbol style.

The following styles are defined:

NoSymbol No Style. The symbol cannot be drawn.

Ellipse Ellipse or circle

Rect Rectangle

Diamond Diamond

Triangle Triangle pointing upwards

DTriangle Triangle pointing downwards

UTriangle Triangle pointing upwards

LTriangle Triangle pointing left

RTriangle Triangle pointing right

Cross Cross (+)

XCross Diagonal cross (X)

HLine Horizontal line

VLine Vertical line

Star1 X combined with +

Star2 Six-pointed star

Hexagon Hexagon

Parameters

s style

12.82.4.14 const QSize& QwtSymbol::size () const [inline]

Return Size.

12.82.4.15 Style QwtSymbol::style () const [inline]

Return Style.

12.83 QwtText Class Reference

A class representing a text.

`#include <qwt_text.h>`

Public Types

- enum TextFormat {
  AutoText = 0,
  PlainText,
  RichText,
  MathMLText,
  TeXText,
  OtherFormat = 100 }
- enum PaintAttribute {
  PaintUsingTextFont = 1,
  PaintUsingTextColor = 2,
  PaintBackground = 4 }
- enum LayoutAttribute { **MinimumLayout** = 1 }

Public Member Functions

- QwtText (const QString &=QString::null, TextFormat textFormat=AutoText)
- QwtText (const QwtText &)
- ~QwtText ()
- QwtText & operator= (const QwtText &)
- int operator== (const QwtText &) const
- int operator!= (const QwtText &) const
- void setText (const QString &, QwtText::TextFormat textFormat=AutoText)
- QString text () const
- bool isNull () const
- bool isEmpty () const
- void setFont (const QFont &)
- QFont font () const
- QFont usedFont (const QFont &) const
- void setRenderFlags (int flags)
- int renderFlags () const
- void setColor (const QColor &)
- QColor color () const
- QColor usedColor (const QColor &) const
- void setBackgroundPen (const QPen &)
- QPen backgroundPen () 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
- int heightForWidth (int width, const QFont &=QFont()) const
- QSize textSize (const QFont &=QFont()) const
- void draw (QPainter *painter, const QRect &rect) const

Static Public Member Functions

- static const QwtTextEngine * textEngine (const QString &text, QwtText::TextFormat=AutoText)
- static const QwtTextEngine * textEngine (QwtText::TextFormat)
- static void setTextEngine (QwtText::TextFormat, QwtTextEngine *)

12.83.1 Detailed Description

A class representing a text. A QwtText is a text including a set of attributes how to render it.

- Format

  A text might include control sequences (f.e tags) describing how to render it. Each format (f.e MathML, TeX, Qt Rich Text) has its own set of control sequences, that can be handles by a QwtTextEngine for this format.

- Background

  A text might have a background, defined by a QPen and QBrush to improve its visibility.

- Font

  A text might have an individual font.

- Color

  A text might have an individual color.

- Render Flags

  Flags from Qt::AlignmentFlag and Qt::TextFlag used like in QPainter::drawText.

See also

QwtTextEngine, QwtTextLabel

12.83.2 Member Enumeration Documentation

12.83.2.1 enum QwtText::LayoutAttribute

Layout Attributes.

The layout attributes affects some aspects of the layout of the text.

- MinimumLayout

  Layout the text without its margins. This mode is useful if a text needs to be aligned accurately, like the tick labels of a scale. If QwtTextEngine::textMargins is not implemented for the format of the text, MinimumLayout has no effect.

12.83.2.2 enum QwtText::PaintAttribute

Paint Attributes.

Font and color and background are optional attributes of a QwtText. The paint attributes hold the information, if they are set.

- PaintUsingTextFont

  The text has an individual font.

- PaintUsingTextColor

  The text has an individual color.

- PaintBackground

  The text has an individual background.

12.83.2.3 enum QwtText::TextFormat

Text format.

The text format defines the QwtTextEngine, that is used to render the text.

- AutoText

  The text format is determined using QwtTextEngine::mightRender for all available text engines in increasing order > PlainText. If none of the text engines can render the text is rendered like PlainText.

- PlainText

  Draw the text as it is, using a QwtPlainTextEngine.

- RichText

  Use the Scribe framework (Qt Rich Text) to render the text.

- MathMLText

  Use a MathML (http://en.wikipedia.org/wiki/MathML) render engine to display the text. The Qwt MathML extension offers such an engine based on the MathML renderer of the Qt solutions package. Unfortunately it is only available for owners of a commercial Qt license.

- TeXText

  Use a TeX (http://en.wikipedia.org/wiki/TeX) render engine to display the text.

- OtherFormat

  The number of text formats can be extended using setTextEngine. Formats >= OtherFormat are not used by Qwt.

See also

QwtTextEngine, setTextEngine()

12.83.3 Constructor & Destructor Documentation

12.83.3.1 QwtText::QwtText (const QString & *text* = `QString::null`, QwtText::TextFormat *textFormat* = `AutoText`)

Constructor

Parameters

text Text content

textFormat Text format

12.83.3.2 QwtText::QwtText (const QwtText & *other*)

Copy constructor.

12.83.3.3 QwtText::~QwtText ()

Destructor.

12.83.4 Member Function Documentation

12.83.4.1 QBrush QwtText::backgroundBrush () const

Returns

Background brush

See also

setBackgroundBrush(), backgroundPen()

12.83.4.2 QPen QwtText::backgroundPen () const

Returns

Background pen

See also

setBackgroundPen(), backgroundBrush()

12.83.4.3 QColor QwtText::color () const

Return the pen color, used for painting the text.

12.83.4.4 void QwtText::draw (QPainter * *painter*, const QRect & *rect*) const

Draw a text into a rectangle

Parameters

painter Painter

rect Rectangle

12.83.4.5 QFont QwtText::font () const

Return the font.

12.83.4.6 int QwtText::heightForWidth (int *width*, const QFont & *defaultFont* = `QFont()`) const

Find the height for a given width

Parameters

defaultFont Font, used for the calculation if the text has no font

width Width

Returns

Calculated height

12.83.4.7 bool QwtText::isEmpty () const `[inline]`

Returns

text().isEmpty()

12.83.4.8 bool QwtText::isNull () const `[inline]`

Returns

text().isNull()

12.83.4.9 int QwtText::operator!= (const QwtText & *other*) const

Relational operator.

12.83.4.10 QwtText & QwtText::operator= (const QwtText & *other*)

Assignment operator.

12.83.4.11 int QwtText::operator== (const QwtText & *other*) const

Relational operator.

12.83.4.12 int QwtText::renderFlags () const

Returns

Render flags

See also

setRenderFlags()

12.83.4.13 void QwtText::setBackgroundBrush (const QBrush & *brush*)

Set the background brush

Parameters

brush Background brush

See also

backgroundBrush(), setBackgroundPen()

12.83.4.14 void QwtText::setBackgroundPen (const QPen & *pen*)

Set the background pen

Parameters

pen Background pen

See also

backgroundPen(), setBackgroundBrush()

12.83.4.15 void QwtText::setColor (const QColor & *color*)

Set the pen color used for painting the text.

Parameters

color Color

Note

Setting the color might have no effect, when the text contains control sequences for setting colors.

12.83.4.16 void QwtText::setFont (const QFont & *font*)

Set the font.

Parameters

font Font

Note

Setting the font might have no effect, when the text contains control sequences for setting fonts.

12.83.4.17 void QwtText::setLayoutAttribute (LayoutAttribute *attribute*, bool *on* = `true`)

Change a layout attribute

Parameters

attribute Layout attribute

on On/Off

See also

testLayoutAttribute()

12.83.4.18 void QwtText::setPaintAttribute (PaintAttribute *attribute*, bool *on* = `true`)

Change a paint attribute

Parameters

attribute Paint attribute

on On/Off

Note

Used by setFont(), setColor(), setBackgroundPen() and setBackgroundBrush()

See also

testPaintAttribute()

12.83.4.19 void QwtText::setRenderFlags (int *renderFlags*)

Change the render flags.

The default setting is Qt::AlignCenter

Parameters

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.

12.83.4.20 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()

12.83.4.21 void QwtText::setTextEngine (QwtText::TextFormat *format*, QwtTextEngine * *engine*) `[static]`

Assign/Replace a text engine for a text format

With setTextEngine it is possible to extend Qwt with other types of text formats.

Owner of a commercial Qt license can build the qwtmathml library, that is based on the MathML renderer, that is included in MML Widget component of the Qt solutions package.

For QwtText::PlainText it is not allowed to assign a engine == NULL.

Parameters

format Text format

engine Text engine

See also

QwtMathMLTextEngine

Warning

Using QwtText::AutoText does nothing.

12.83.4.22 bool QwtText::testLayoutAttribute (LayoutAttribute *attribute*) const

Test a layout attribute

Parameters

attribute Layout attribute

Returns

true, if attribute is enabled

See also

setLayoutAttribute()

12.83.4.23 bool QwtText::testPaintAttribute (PaintAttribute *attribute*) const

Test a paint attribute

Parameters

attribute Paint attribute

Returns

true, if attribute is enabled

See also

setPaintAttribute()

12.83.4.24 QString QwtText::text () const

Return the text.

See also

setText()

12.83.4.25 const QwtTextEngine * QwtText::textEngine (QwtText::TextFormat *format*) [static]

Find the text engine for a text format.

textEngine can be used to find out if a text format is supported. F.e, if one wants to use MathML labels, the MathML renderer from the commercial Qt solutions package might be required, that is not available in Qt Open Source Edition environments.

Parameters

format Text format

Returns

The text engine, or NULL if no engine is available.

12.83.4.26 const QwtTextEngine * QwtText::textEngine (const QString & *text*, QwtText::TextFormat *format* = `AutoText`) `[static]`

Find the text engine for a text format

In case of QwtText::AutoText the first text engine (beside QwtPlainTextEngine) is returned, where QwtTextEngine::mightRender returns true. If there is none QwtPlainTextEngine is returnd.

If no text engine is registered for the format QwtPlainTextEngine is returnd.

Parameters

text Text, needed in case of AutoText

format Text format

12.83.4.27 QSize QwtText::textSize (const QFont & *defaultFont* = `QFont()`) const

Find the height for a given width

Parameters

defaultFont Font, used for the calculation if the text has no font

Returns

Calculated height

Returns the size, that is needed to render text

Parameters

defaultFont Font of the text

Returns

Caluclated size

12.83.4.28 QColor QwtText::usedColor (const QColor & *defaultColor*) const

Return the color of the text, if it has one. Otherwise return defaultColor.

Parameters

defaultColor Default color

See also

setColor(), color(), PaintAttributes

12.83.4.29 QFont QwtText::usedFont (const QFont & *defaultFont*) const

Return the font of the text, if it has one. Otherwise return defaultFont.

Parameters

defaultFont Default font

See also

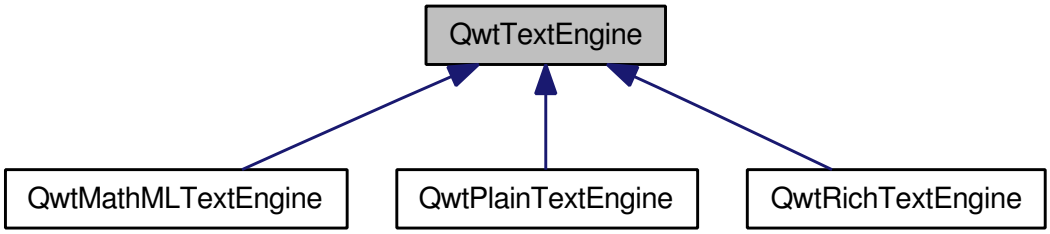
setFont(), font(), PaintAttributes

12.84 QwtTextEngine Class Reference

Abstract base class for rendering text strings.

`#include <qwt_text_engine.h>`

Inheritance diagram for QwtTextEngine:



Public Member Functions

- virtual ~QwtTextEngine ()
- virtual int heightForWidth (const QFont &font, int flags, const QString &text, int width) const =0
- virtual QSize 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, int &left, int &right, int &top, int &bottom) const =0
- virtual void draw (QPainter *painter, const QRect &rect, int flags, const QString &text) const =0

Protected Member Functions

- QwtTextEngine ()

12.84.1 Detailed Description

Abstract base class for rendering text strings. A text engine is responsible for rendering texts for a specific text format. They are used by QwtText to render a text.

QwtPlainTextEngine and QwtRichTextEngine are part of the Qwt library.

QwtMathMLTextEngine can be found in Qwt MathML extension, that needs the MathML renderer of the Qt solutions package. Unfortunately it is only available with a commercial Qt license.

See also

QwtText::setTextEngine()

12.84.2 Constructor & Destructor Documentation

12.84.2.1 QwtTextEngine::~QwtTextEngine () [virtual]

Destructor.

12.84.2.2 QwtTextEngine::QwtTextEngine () [protected]

Constructor.

12.84.3 Member Function Documentation

12.84.3.1 virtual void QwtTextEngine::draw (QPainter * *painter*, const QRect & *rect*, int *flags*, const QString & *text*) const [pure virtual]

Draw the text in a clipping rectangle

Parameters

painter Painter

rect Clipping rectangle

flags Bitwise OR of the flags like in for QPainter::drawText

text Text to be rendered

Implemented in QwtPlainTextEngine, QwtRichTextEngine, and QwtMathMLTextEngine.

12.84.3.2 virtual int QwtTextEngine::heightForWidth (const QFont & *font*, int *flags*, const QString & *text*, int *width*) const [pure virtual]

Find the height for a given width

Parameters

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 QwtPlainTextEngine, QwtRichTextEngine, and QwtMathMLTextEngine.

12.84.3.3 virtual bool QwtTextEngine::mightRender (const QString & *text*) const [pure virtual]

Test if a string can be rendered by this text engine

Parameters

text Text to be tested

Returns

true, if it can be rendered

Implemented in QwtPlainTextEngine, QwtRichTextEngine, and QwtMathMLTextEngine.

12.84.3.4 virtual void QwtTextEngine::textMargins (const QFont & *font*, const QString & *text*, int & *left*, int & *right*, int & *top*, int & *bottom*) const [pure virtual]

Return margins around the texts

The textSize might include margins around the text, like QFontMetrics::descent. In situations where texts need to be aligend 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 QwtPlainTextEngine, QwtRichTextEngine, and QwtMathMLTextEngine.

12.84.3.5 virtual QSize QwtTextEngine::textSize (const QFont & *font*, int *flags*, const QString & *text*) const [pure virtual]

Returns the size, that is needed to render text

Parameters

font Font of the text

flags Bitwise OR of the flags like in for QPainter::drawText

text Text to be rendered

Returns

Caluclated size

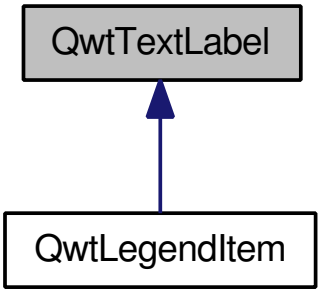
Implemented in QwtPlainTextEngine, QwtRichTextEngine, and QwtMathMLTextEngine.

12.85 QwtTextLabel Class Reference

A Widget which displays a QwtText.

`#include <qwt_text_label.h>`

Inheritance diagram for QwtTextLabel:



Public Slots

- void setText (const QString &, QwtText::TextFormat textFormat=QwtText::AutoText)
- virtual void setText (const QwtText &)
- void clear ()

Public Member Functions

- QwtTextLabel (QWidget *parent=NULL)
- QwtTextLabel (const QwtText &, QWidget *parent=NULL)
- virtual ~QwtTextLabel ()
- const QwtText & text () const
- int indent () const
- void setIndent (int)
- int margin () const
- void setMargin (int)
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- virtual int heightForWidth (int) const
- QRect textRect () const

Protected Member Functions

- virtual void paintEvent (QPaintEvent *e)
- virtual void drawContents (QPainter *)
- virtual void drawText (QPainter *, const QRect &)

12.85.1 Detailed Description

A Widget which displays a QwtText.

12.85.2 Constructor & Destructor Documentation

12.85.2.1 QwtTextLabel::QwtTextLabel (QWidget * *parent* = NULL) [explicit]

Constructs an empty label.

Parameters

parent Parent widget

12.85.2.2 QwtTextLabel::QwtTextLabel (const QwtText & *text*, QWidget * *parent* = NULL) [explicit]

Constructs a label that displays the text, text

Parameters

parent Parent widget

text Text

12.85.2.3 QwtTextLabel::~QwtTextLabel () [virtual]

Destructor.

12.85.3 Member Function Documentation

12.85.3.1 void QwtTextLabel::clear () [slot]

Clear the text and all QwtText attributes.

12.85.3.2 void QwtTextLabel::drawContents (QPainter * *painter*) [protected, virtual]

Redraw the text and focus indicator.

12.85.3.3 void QwtTextLabel::drawText (QPainter * *painter*, const QRect & *textRect*) [protected, virtual]

Redraw the text.

Reimplemented in QwtLegendItem.

12.85.3.4 int QwtTextLabel::heightForWidth (int *width*) const [virtual]

Returns the preferred height for this widget, given the width.

Parameters

width Width

12.85.3.5 int QwtTextLabel::indent () const

Return label's text indent in pixels.

12.85.3.6 int QwtTextLabel::margin () const

Return label's text indent in pixels.

12.85.3.7 QSize QwtTextLabel::minimumSizeHint () const [virtual]

Return a minimum size hint.

12.85.3.8 void QwtTextLabel::paintEvent (QPaintEvent * *event*) [protected, virtual]

Qt paint event

Parameters

event Paint event

Reimplemented in QwtLegendItem.

12.85.3.9 void QwtTextLabel::setIndent (int *indent*)

Set label's text indent in pixels

Parameters

indent Indentation in pixels

12.85.3.10 void QwtTextLabel::setMargin (int *margin*)

Set label's margin in pixels

Parameters

margin Margin in pixels

12.85.3.11 void QwtTextLabel::setText (const QwtText & *text*) [virtual, slot]

Change the label's text

Parameters

text New text

Reimplemented in QwtLegendItem.

12.85.3.12 void QwtTextLabel::setText (const QString & *text*, QwtText::TextFormat *textFormat* = QwtText::AutoText) [slot]

Change the label's text, keeping all other QwtText attributes

Parameters

text New text

textFormat Format of text

See also

QwtText

12.85.3.13 QSize QwtTextLabel::sizeHint () const [virtual]

Return label's margin in pixels.

Reimplemented in QwtLegendItem.

12.85.3.14 const QwtText & QwtTextLabel::text () const

Return the text.

12.85.3.15 QRect QwtTextLabel::textRect () const

Calculate the rect for the text in widget coordinates

Returns

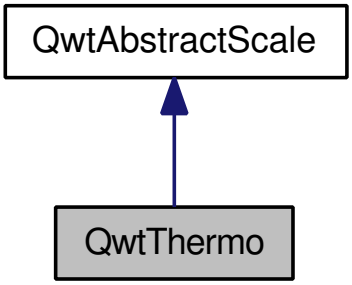
Text rect

12.86 QwtThermo Class Reference

The Thermometer Widget.

`#include <qwt_thermo.h>`

Inheritance diagram for QwtThermo:



Public Types

- enum **ScalePos** {
  NoScale,
  LeftScale,
  RightScale,
  TopScale,
  BottomScale }

Public Slots

- void setValue (double val)

Public Member Functions

- QwtThermo (QWidget ∗parent=NULL)
- virtual ∼QwtThermo ()
- void setOrientation (Qt::Orientation o, ScalePos s)
- void setScalePosition (ScalePos s)
- ScalePos scalePosition () const
- void setBorderWidth (int w)
- int borderWidth () const
- void setFillBrush (const QBrush &b)
- const QBrush & fillBrush () const
- void setFillColor (const QColor &c)
- const QColor & fillColor () const
- void setAlarmBrush (const QBrush &b)
- const QBrush & alarmBrush () const
- void setAlarmColor (const QColor &c)
- const QColor & alarmColor () const
- void setAlarmLevel (double v)
- double alarmLevel () const
- void setAlarmEnabled (bool tf)
- bool alarmEnabled () const
- void setPipeWidth (int w)

- int pipeWidth () const
- void setMaxValue (double v)
- double maxValue () const
- void setMinValue (double v)
- double minValue () const
- double value () const
- void setRange (double vmin, double vmax, bool lg=false)
- void setMargin (int m)
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const
- void setScaleDraw (QwtScaleDraw *)
- const QwtScaleDraw * scaleDraw () const

Protected Member Functions

- void draw (QPainter *p, const QRect &update_rect)
- void drawThermo (QPainter *p)
- void layoutThermo (bool update=true)
- virtual void scaleChange ()
- virtual void fontChange (const QFont &oldFont)
- virtual void paintEvent (QPaintEvent *e)
- virtual void resizeEvent (QResizeEvent *e)
- QwtScaleDraw * scaleDraw ()

12.86.1 Detailed Description

The Thermometer Widget. QwtThermo is a widget which displays a value in an interval. It supports:

- a horizontal or vertical layout;
- a range;
- a scale;
- an alarm level.

By default, the scale and range run over the same interval of values. QwtAbstractScale::setScale() changes the interval of the scale and allows easy conversion between physical units.

The example shows how to make the scale indicate in degrees Fahrenheit and to set the value in degrees Kelvin:

```
#include <qapplication.h>
#include <qwt_thermo.h>

double Kelvin2Fahrenheit(double kelvin)
{
    // see http://en.wikipedia.org/wiki/Kelvin
    return 1.8*kelvin - 459.67;
}

int main(int argc, char **argv)
{
    const double minKelvin = 0.0;
    const double maxKelvin = 500.0;
```

```
    QApplication a(argc, argv);
    QwtThermo t;
    t.setRange(minKelvin, maxKelvin);
    t.setScale(Kelvin2Fahrenheit(minKelvin), Kelvin2Fahrenheit(maxKelvin));
    // set the value in Kelvin but the scale displays in Fahrenheit
    // 273.15 Kelvin = 0 Celsius = 32 Fahrenheit
    t.setValue(273.15);
    a.setMainWidget(&t);
    t.show();
    return a.exec();
}
```

12.86.2 Constructor & Destructor Documentation

12.86.2.1 QwtThermo::QwtThermo (QWidget * *parent* = NULL) [explicit]

Constructor

Parameters

parent Parent widget

12.86.2.2 QwtThermo::~QwtThermo () [virtual]

Destructor.

12.86.3 Member Function Documentation

12.86.3.1 const QBrush & QwtThermo::alarmBrush () const

Return the liquid brush above the alarm threshold.

See also

setAlarmBrush()

12.86.3.2 const QColor & QwtThermo::alarmColor () const

Return the liquid color above the alarm threshold.

12.86.3.3 bool QwtThermo::alarmEnabled () const

Return if the alarm threshold is enabled or disabled.

12.86.3.4 double QwtThermo::alarmLevel () const

Return the alarm threshold.

See also

setAlarmLevel()

12.86.3.5 int QwtThermo::borderWidth () const

Return the border width of the thermometer pipe.

See also

setBorderWidth()

12.86.3.6 void QwtThermo::draw (QPainter * *painter*, const QRect & *rect*) [protected]

Draw the whole QwtThermo.

Parameters

painter Painter

rect Update rectangle

12.86.3.7 void QwtThermo::drawThermo (QPainter * *painter*) [protected]

Redraw the liquid in thermometer pipe.

Parameters

painter Painter

12.86.3.8 const QBrush & QwtThermo::fillBrush () const

Return the liquid brush.

See also

setFillBrush()

12.86.3.9 const QColor & QwtThermo::fillColor () const

Return the liquid color.

See also

setFillColor()

12.86.3.10 void QwtThermo::fontChange (const QFont & *oldFont*) [protected, virtual]

Notify a font change.

12.86.3.11 void QwtThermo::layoutThermo (bool *update_geometry* = `true`) `[protected]`

Recalculate the QwtThermo geometry and layout based on the QwtThermo::rect() and the fonts.

Parameters

update_geometry notify the layout system and call update to redraw the scale

12.86.3.12 double QwtThermo::maxValue () const

Return the maximum value.

12.86.3.13 QSize QwtThermo::minimumSizeHint () const `[virtual]`

Return a minimum size hint.

Warning

The return value depends on the font and the scale.

See also

sizeHint()

12.86.3.14 double QwtThermo::minValue () const

Return the minimum value.

12.86.3.15 void QwtThermo::paintEvent (QPaintEvent * *event*) `[protected, virtual]`

Qt paint event. event Paint event

12.86.3.16 int QwtThermo::pipeWidth () const

Return the width of the pipe.

See also

setPipeWidth()

12.86.3.17 void QwtThermo::resizeEvent (QResizeEvent * *e*) `[protected, virtual]`

Qt resize event handler.

12.86.3.18 void QwtThermo::scaleChange () [protected, virtual]

Notify a scale change.

Reimplemented from QwtAbstractScale.

12.86.3.19 QwtScaleDraw * QwtThermo::scaleDraw () [protected]

Returns

the scale draw of the thermo

See also

setScaleDraw()

12.86.3.20 const QwtScaleDraw * QwtThermo::scaleDraw () const

Returns

the scale draw of the thermo

See also

setScaleDraw()

12.86.3.21 QwtThermo::ScalePos QwtThermo::scalePosition () const

Return the scale position.

See also

setScalePosition()

12.86.3.22 void QwtThermo::setAlarmBrush (const QBrush & *brush*)

Specify the liquid brush above the alarm threshold.

Parameters

brush New brush. The default is solid white.

See also

alarmBrush()

12.86.3.23 void QwtThermo::setAlarmColor (const QColor & *c*)

Specify the liquid color above the alarm threshold.

Parameters

c New color. The default is white.

12.86.3.24 void QwtThermo::setAlarmEnabled (bool *tf*)

Enable or disable the alarm threshold.

Parameters

tf true (disabled) or false (enabled)

12.86.3.25 void QwtThermo::setAlarmLevel (double *level*)

Specify the alarm threshold.

Parameters

level Alarm threshold

See also

alarmLevel()

12.86.3.26 void QwtThermo::setBorderWidth (int *width*)

Set the border width of the pipe.

Parameters

width Border width

See also

borderWidth()

12.86.3.27 void QwtThermo::setFillBrush (const QBrush & *brush*)

Change the brush of the liquid.

Parameters

brush New brush. The default brush is solid black.

See also

fillBrush()

12.86.3.28 void QwtThermo::setFillColor (const QColor & *c*)

Change the color of the liquid.

Parameters

c New color. The default color is black.

See also

fillColor()

12.86.3.29 void QwtThermo::setMargin (int *m*)

Specify the distance between the pipe's endpoints and the widget's border.

The margin is used to leave some space for the scale labels. If a large font is used, it is advisable to adjust the margins.

Parameters

m New Margin. The default values are 10 for horizontal orientation and 20 for vertical orientation.

Warning

The margin has no effect if the scale is disabled.
This function is a NOOP because margins are determined automatically.

12.86.3.30 void QwtThermo::setMaxValue (double *max*)

Set the maximum value.

Parameters

max Maximum value

See also

maxValue(), setMinValue()

12.86.3.31 void QwtThermo::setMinValue (double *min*)

Set the minimum value.

Parameters

min Minimum value

See also

minValue(), setMaxValue()

12.86.3.32 void QwtThermo::setOrientation (Qt::Orientation *o*, ScalePos *s*)

Set the thermometer orientation and the scale position.

The scale position NoScale disables the scale.

Parameters

o orientation. Possible values are Qt::Horizontal and Qt::Vertical. The default value is Qt::Vertical.

s Position of the scale. The default value is NoScale.

A valid combination of scale position and orientation is enforced:

- a horizontal thermometer can have the scale positions TopScale, BottomScale or NoScale;
- a vertical thermometer can have the scale positions LeftScale, RightScale or NoScale;
- an invalid scale position will default to NoScale.

See also

setScalePosition()

12.86.3.33 void QwtThermo::setPipeWidth (int *width*)

Change the width of the pipe.

Parameters

width Width of the pipe

See also

pipeWidth()

12.86.3.34 void QwtThermo::setRange (double *vmin*, double *vmax*, bool *logarithmic* = `false`)

Set the range.

Parameters

vmin value corresponding lower or left end of the thermometer

vmax value corresponding to the upper or right end of the thermometer

logarithmic logarithmic mapping, true or false

12.86.3.35 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().

12.86.3.36 void QwtThermo::setScalePosition (ScalePos *scalePos*)

Change the scale position (and thermometer orientation).

Parameters

scalePos Position of the scale.

A valid combination of scale position and orientation is enforced:

- if the new scale position is LeftScale or RightScale, the scale orientation will become Qt::Vertical;
- if the new scale position is BottomScale or TopScale, the scale orientation will become Qt::Horizontal;
- if the new scale position is NoScale, the scale orientation will not change.

See also

setOrientation(), scalePosition()

12.86.3.37 void QwtThermo::setValue (double *value*) [slot]

Set the current value.

Parameters

value New Value

See also

value()

12.86.3.38 QSize QwtThermo::sizeHint () const [virtual]

Returns

the minimum size hint

See also

minimumSizeHint()

12.86.3.39 double QwtThermo::value () const

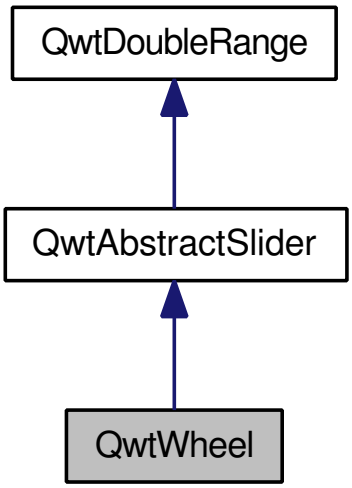
Return the value.

12.87 QwtWheel Class Reference

The Wheel Widget.

`#include <qwt_wheel.h>`

Inheritance diagram for QwtWheel:



Public Member Functions

- QwtWheel (QWidget *parent=NULL)
- virtual ~QwtWheel ()
- virtual void setOrientation (Qt::Orientation)
- double totalAngle () const
- double viewAngle () const
- int tickCnt () const
- int internalBorder () const
- double mass () const
- void setTotalAngle (double angle)
- void setTickCnt (int cnt)
- void setViewAngle (double angle)

- void setInternalBorder (int width)
- void setMass (double val)
- void setWheelWidth (int w)
- virtual QSize sizeHint () const
- virtual QSize minimumSizeHint () const

Protected Member Functions

- virtual void resizeEvent (QResizeEvent ∗e)
- virtual void paintEvent (QPaintEvent ∗e)
- void layoutWheel (bool update=true)
- void draw (QPainter ∗, const QRect &)
- void drawWheel (QPainter ∗, const QRect &)
- void drawWheelBackground (QPainter ∗, const QRect &)
- void setColorArray ()
- virtual void valueChange ()
- virtual void paletteChange (const QPalette &)
- virtual double getValue (const QPoint &)
- virtual void getScrollMode (const QPoint &, int &scrollMode, int &direction)

12.87.1 Detailed Description

The Wheel Widget. The wheel widget can be used to change values over a very large range in very small steps. Using the setMass member, it can be configured as a flywheel.

See also

The radio example.

12.87.2 Constructor & Destructor Documentation

12.87.2.1 QwtWheel::QwtWheel (QWidget ∗ *parent* = `NULL`) `[explicit]`

Constructor.

12.87.2.2 QwtWheel::∼QwtWheel () `[virtual]`

Destructor.

12.87.3 Member Function Documentation

12.87.3.1 void QwtWheel::draw (QPainter ∗ *painter*, const QRect &) `[protected]`

Redraw panel and wheel

Parameters

painter Painter

12.87.3.2 void QwtWheel::drawWheel (QPainter * *painter*, const QRect & *r*) [protected]

Redraw the wheel.

Parameters

painter painter

r contents rectangle

12.87.3.3 void QwtWheel::drawWheelBackground (QPainter * *painter*, const QRect & *r*) [protected]

Draw the Wheel's background gradient

Parameters

painter Painter

r Bounding rectangle

12.87.3.4 void QwtWheel::getScrollMode (const QPoint & *p*, int & *scrollMode*, int & *direction*) [protected, virtual]

Determine the scrolling mode and direction corresponding to a specified point.

Parameters

p point

scrollMode scrolling mode

direction direction

Implements QwtAbstractSlider.

12.87.3.5 double QwtWheel::getValue (const QPoint & *p*) [protected, virtual]

Determine the value corresponding to a specified point.

Implements QwtAbstractSlider.

12.87.3.6 int QwtWheel::internalBorder () const

Returns

Internal border width of the wheel.

See also

setInternalBorder()

12.87.3.7 void QwtWheel::layoutWheel (bool *update* = `true`) `[protected]`

Recalculate the slider's geometry and layout based on.

12.87.3.8 double QwtWheel::mass () const `[virtual]`

Returns

mass

Reimplemented from QwtAbstractSlider.

12.87.3.9 QSize QwtWheel::minimumSizeHint () const `[virtual]`

Return a minimum size hint.

Warning

The return value is based on the wheel width.

12.87.3.10 void QwtWheel::paintEvent (QPaintEvent * *e*) `[protected, virtual]`

Qt Paint Event.

12.87.3.11 void QwtWheel::paletteChange (const QPalette &) `[protected, virtual]`

Call update() when the palette changes.

12.87.3.12 void QwtWheel::resizeEvent (QResizeEvent * *e*) `[protected, virtual]`

Qt Resize Event.

12.87.3.13 void QwtWheel::setColorArray () `[protected]`

Set up the color array for the background pixmap.

12.87.3.14 void QwtWheel::setInternalBorder (int *w*)

Set the internal border width of the wheel.

The internal border must not be smaller than 1 and is limited in dependence on the wheel's size. Values outside the allowed range will be clipped.

The internal border defaults to 2.

Parameters

w border width

See also

internalBorder()

12.87.3.15 void QwtWheel::setMass (double *val*) [virtual]

Set the mass of the wheel.

Assigning a mass turns the wheel into a flywheel.

Parameters

val the wheel's mass

Reimplemented from QwtAbstractSlider.

12.87.3.16 void QwtWheel::setOrientation (Qt::Orientation *o*) [virtual]

Set the wheel's orientation.

Parameters

o Orientation. Allowed values are Qt::Horizontal and Qt::Vertical. Defaults to Qt::Horizontal.

See also

QwtAbstractSlider::orientation()

Reimplemented from QwtAbstractSlider.

12.87.3.17 void QwtWheel::setTickCnt (int *cnt*)

Adjust the number of grooves in the wheel's surface.

The number of grooves is limited to 6 <= cnt <= 50. Values outside this range will be clipped. The default value is 10.

Parameters

cnt Number of grooves per 360 degrees

See also

tickCnt()

12.87.3.18 void QwtWheel::setTotalAngle (double *angle*)

Set the total angle which the wheel can be turned.

One full turn of the wheel corresponds to an angle of 360 degrees. A total angle of n∗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()

12.87.3.19 void QwtWheel::setViewAngle (double *angle*)

Specify the visible portion of the wheel.

You may use this function for fine-tuning the appearance of the wheel. The default value is 175 degrees. The value is limited from 10 to 175 degrees.

Parameters

angle Visible angle in degrees

See also

viewAngle(), setTotalAngle()

12.87.3.20 void QwtWheel::setWheelWidth (int *w*)

Set the width of the wheel.

Corresponds to the wheel height for horizontal orientation, and the wheel width for vertical orientation.

Parameters

w the wheel's width

12.87.3.21 QSize QwtWheel::sizeHint () const [virtual]

Returns

a size hint

12.87.3.22 int QwtWheel::tickCnt () const

Returns

Number of grooves in the wheel's surface.

See also

setTickCnt()

12.87.3.23 double QwtWheel::totalAngle () const

Returns

Total angle which the wheel can be turned.

See also

setTotalAngle()

12.87.3.24 void QwtWheel::valueChange () [protected, virtual]

Notify value change.

Reimplemented from QwtAbstractSlider.

12.87.3.25 double QwtWheel::viewAngle () const

Returns

Visible portion of the wheel

See also

setViewAngle(), totalAngle()

Index