

# Qwt Polar User's Guide

1.1.1

Generated by Doxygen 1.8.5

Fri Sep 19 2014 12:28:25

## Contents

<b>1</b>	<b>QwtPolar - A Qwt/Qt Polar Plot Library</b>	<b>2</b>
1.1	License	2
1.2	Platforms	2
1.3	Downloads	2
1.4	Support	2
1.5	Related Projects	2
1.6	Donations	2
1.7	Credits:	3
<b>2</b>	<b>Qwt License, Version 1.0</b>	<b>3</b>
<b>3</b>	<b>Hierarchical Index</b>	<b>9</b>
3.1	Class Hierarchy	9
<b>4</b>	<b>Class Index</b>	<b>10</b>
4.1	Class List	10
<b>5</b>	<b>Class Documentation</b>	<b>11</b>
5.1	QwtPolarCanvas Class Reference	11
5.1.1	Detailed Description	12
5.1.2	Member Enumeration Documentation	12
5.1.3	Member Function Documentation	12
5.2	QwtPolarCurve Class Reference	15
5.2.1	Detailed Description	16
5.2.2	Member Enumeration Documentation	16
5.2.3	Constructor & Destructor Documentation	17
5.2.4	Member Function Documentation	17
5.3	QwtPolarFitter Class Reference	22
5.3.1	Detailed Description	23
5.3.2	Constructor & Destructor Documentation	23
5.3.3	Member Function Documentation	23
5.4	QwtPolarGrid Class Reference	24
5.4.1	Detailed Description	26
5.4.2	Member Enumeration Documentation	26
5.4.3	Constructor & Destructor Documentation	27
5.4.4	Member Function Documentation	27
5.5	QwtPolarItem Class Reference	34
5.5.1	Detailed Description	35
5.5.2	Member Enumeration Documentation	36
5.5.3	Constructor & Destructor Documentation	36

5.5.4	Member Function Documentation	37
5.6	QwtPolarItemDict Class Reference	42
5.6.1	Detailed Description	43
5.6.2	Constructor & Destructor Documentation	43
5.6.3	Member Function Documentation	44
5.7	QwtPolarLayout Class Reference	45
5.7.1	Detailed Description	46
5.7.2	Member Enumeration Documentation	46
5.7.3	Member Function Documentation	46
5.8	QwtPolarMagnifier Class Reference	48
5.8.1	Detailed Description	49
5.8.2	Constructor & Destructor Documentation	49
5.8.3	Member Function Documentation	49
5.9	QwtPolarMarker Class Reference	50
5.9.1	Detailed Description	51
5.9.2	Member Function Documentation	52
5.10	QwtPolarPanner Class Reference	53
5.10.1	Detailed Description	54
5.10.2	Member Function Documentation	54
5.11	QwtPolarPicker Class Reference	55
5.11.1	Detailed Description	56
5.11.2	Constructor & Destructor Documentation	57
5.11.3	Member Function Documentation	58
5.12	QwtPolarPlot Class Reference	61
5.12.1	Detailed Description	64
5.12.2	Member Enumeration Documentation	64
5.12.3	Constructor & Destructor Documentation	64
5.12.4	Member Function Documentation	65
5.13	QwtPolarRenderer Class Reference	76
5.13.1	Detailed Description	76
5.13.2	Constructor & Destructor Documentation	76
5.13.3	Member Function Documentation	77
5.14	QwtPolarSpectrogram Class Reference	80
5.14.1	Detailed Description	81
5.14.2	Member Enumeration Documentation	81
5.14.3	Member Function Documentation	81

# 1 QwtPolar - A Qwt/Qt Polar Plot Library

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

## 1.1 License

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

## 1.2 Platforms

QwtPolar depends on the [Qt](#) and [Qwt](#) frameworks and might be usable in all environments supported by [Qt](#). It is compatible with [Qt](#)  $\geq 4.4$  and [Qwt](#)  $\geq 6.1$ .

## 1.3 Downloads

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

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

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

For getting a development snapshot from the SVN repository:

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

## 1.4 Support

- Mailing list

QwtPolar doesn't have its own mailing list, but you can ask on the Qwt [mailing list](#).

If you prefer newsgroups use the mail to news gateway of [Gmane](#).

- Forum

[Qt Centre](#) is a great resource for Qt related questions. It has a sub forum, that is dedicated to Qwt related questions.

- Individual support

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

## 1.5 Related Projects

[Qwt](#), Qt Widgets for Technical Applications.

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

## 1.6 Donations

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

## 1.7 Credits:

### Authors:

Uwe Rathmann

### Project admin:

Uwe Rathmann <[rathmann@users.sourceforge.net](mailto: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 Hierarchical Index

### 3.1 Class Hierarchy

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

QFrame	
<b>QwtPolarCanvas</b>	<b>11</b>
<b>QwtPolarPlot</b>	<b>61</b>
QObject	
<b>QwtPolarRenderer</b>	<b>76</b>
QwtCurveFitter	
<b>QwtPolarFitter</b>	<b>22</b>
QwtMagnifier	
<b>QwtPolarMagnifier</b>	<b>48</b>
QwtPanner	
<b>QwtPolarPanner</b>	<b>53</b>
QwtPicker	
<b>QwtPolarPicker</b>	<b>55</b>
<b>QwtPolarItem</b>	<b>34</b>
<b>QwtPolarCurve</b>	<b>15</b>

<b>QwtPolarGrid</b>	<b>24</b>
<b>QwtPolarMarker</b>	<b>50</b>
<b>QwtPolarSpectrogram</b>	<b>80</b>
<b>QwtPolarItemDict</b>	<b>42</b>
<b>QwtPolarPlot</b>	<b>61</b>
<b>QwtPolarLayout</b>	<b>45</b>

## 4 Class Index

### 4.1 Class List

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

<b>QwtPolarCanvas</b> Canvas of a <b>QwtPolarPlot</b>	<b>11</b>
<b>QwtPolarCurve</b> An item, that represents a series of points	<b>15</b>
<b>QwtPolarFitter</b> A simple curve fitter for polar points	<b>22</b>
<b>QwtPolarGrid</b> An item which draws scales and grid lines on a polar plot	<b>24</b>
<b>QwtPolarItem</b> Base class for items on a polar plot	<b>34</b>
<b>QwtPolarItemDict</b> A dictionary for polar plot items	<b>42</b>
<b>QwtPolarLayout</b> Layout class for <b>QwtPolarPlot</b>	<b>45</b>
<b>QwtPolarMagnifier</b> <b>QwtPolarMagnifier</b> provides zooming, by magnifying in steps	<b>48</b>
<b>QwtPolarMarker</b> A class for drawing markers	<b>50</b>
<b>QwtPolarPanner</b> <b>QwtPolarPanner</b> provides panning of a polar plot canvas	<b>53</b>
<b>QwtPolarPicker</b> <b>QwtPolarPicker</b> provides selections on a plot canvas	<b>55</b>
<b>QwtPolarPlot</b> A plotting widget, displaying a polar coordinate system	<b>61</b>
<b>QwtPolarRenderer</b> Renderer for exporting a polar plot to a document, a printer or anything else, that is supported by QPainter/QPaintDevice	<b>76</b>

**QwtPolarSpectrogram**

An item, which displays a spectrogram

80

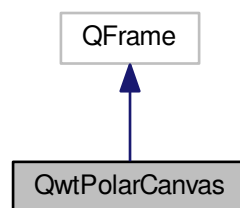
## 5 Class Documentation

### 5.1 QwtPolarCanvas Class Reference

Canvas of a [QwtPolarPlot](#).

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

Inheritance diagram for QwtPolarCanvas:



#### Public Types

- enum [PaintAttribute](#) { [BackingStore](#) = 0x01 }
  - typedef QFlags< [PaintAttribute](#) > [PaintAttributes](#)
- Paint attributes.*

#### Public Member Functions

- [QwtPolarCanvas](#) ([QwtPolarPlot](#) \*)
  - virtual [~QwtPolarCanvas](#) ()
  - [QwtPolarPlot](#) \* [plot](#) ()
  - const [QwtPolarPlot](#) \* [plot](#) () const
  - void [setPaintAttribute](#) ([PaintAttribute](#), bool on=true)
  - bool [testPaintAttribute](#) ([PaintAttribute](#)) const
  - const QPixmap \* [backingStore](#) () const
  - void [invalidateBackingStore](#) ()
  - QwtPointPolar [invTransform](#) (const QPoint &) const
  - QPoint [transform](#) (const QwtPointPolar &) const
- Constructor.*
- Destructor.*
- Changing the paint attributes.*
- Invalidate the internal backing store.*

## Protected Member Functions

- virtual void [paintEvent](#) (QPaintEvent \*)
- virtual void [resizeEvent](#) (QResizeEvent \*)

### 5.1.1 Detailed Description

Canvas of a [QwtPolarPlot](#).

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

#### Note

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

#### See Also

[QwtPolarPlot](#)

### 5.1.2 Member Enumeration Documentation

#### 5.1.2.1 enum [QwtPolarCanvas::PaintAttribute](#)

Paint attributes.

The default setting enables BackingStore

#### See Also

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

#### Enumerator

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

### 5.1.3 Member Function Documentation

#### 5.1.3.1 const QPixmap \* [QwtPolarCanvas::backingStore](#) ( ) const

##### Returns

Backing store, might be null

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

Translate a point from widget into plot coordinates

##### Parameters

<i>pos</i>	Point in widget coordinates of the plot canvas
------------	------------------------------------------------

##### Returns

Point in plot coordinates

#### See Also

[transform\(\)](#)

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

Paint event

## Parameters

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

5.1.3.4 **QwtPolarPlot \* QwtPolarCanvas::plot ( )**

## Returns

Parent plot widget

5.1.3.5 **const QwtPolarPlot \* QwtPolarCanvas::plot ( ) const**

## Returns

Parent plot widget

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

Resize event

## Parameters

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

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

Changing the paint attributes.

## Parameters

<i>attribute</i>	Paint attribute
<i>on</i>	On/Off

The default setting enables BackingStore

## See Also

[testPaintAttribute\(\)](#), [paintCache\(\)](#)

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

Test whether a paint attribute is enabled

## Parameters

<i>attribute</i>	Paint attribute
------------------	-----------------

## Returns

true if the attribute is enabled

## See Also

[setPaintAttribute\(\)](#)

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

Translate a point from plot into widget coordinates



## Parameters

<i>polarPos</i>	Point in plot coordinates
-----------------	---------------------------

## Returns

Point in widget coordinates

## See Also

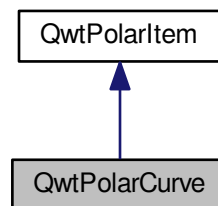
[transform\(\)](#)

## 5.2 QwtPolarCurve Class Reference

An item, that represents a series of points.

```
#include <qwt_polar_curve.h>
```

Inheritance diagram for QwtPolarCurve:



## Public Types

- enum [CurveStyle](#) { [NoCurve](#), [Lines](#), [UserCurve](#) = 100 }
- enum [LegendAttribute](#) { [LegendShowLine](#) = 0x01, [LegendShowSymbol](#) = 0x02 }  
*Attributes how to represent the curve on the legend.*
- typedef QFlags< [LegendAttribute](#) > [LegendAttributes](#)  
*Legend attributes.*

## Public Member Functions

- [QwtPolarCurve](#) ()  
*Constructor.*
- [QwtPolarCurve](#) (const QwtText &[title](#))
- [QwtPolarCurve](#) (const QString &[title](#))
- virtual [~QwtPolarCurve](#) ()  
*Destructor.*
- virtual int [rtti](#) () const
- void [setLegendAttribute](#) ([LegendAttribute](#), bool on=true)
- bool [testLegendAttribute](#) ([LegendAttribute](#)) const  
*Test if a legend attribute is enables.*

- void [setData](#) (QwtSeriesData< QwtPointPolar > \*[data](#))
- const QwtSeriesData  
    < QwtPointPolar > \* [data](#) () const
- size\_t [dataSize](#) () const
- QwtPointPolar [sample](#) (int i) const
- void [setPen](#) (const QPen &)  
    *Assign a pen.*
- const QPen & [pen](#) () const
- void [setStyle](#) (CurveStyle style)
- CurveStyle style () const
- void [setSymbol](#) (QwtSymbol \*)  
    *Assign a symbol.*
- const QwtSymbol \* [symbol](#) () const
- void [setCurveFitter](#) (QwtCurveFitter \*)  
    *Insert a curve fitter.*
- QwtCurveFitter \* [curveFitter](#) () const
- virtual void [draw](#) (QPainter \*p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const  
    QPointF &pole, double radius, const QRectF &canvasRect) const
- virtual void [draw](#) (QPainter \*p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const  
    QPointF &pole, int from, int to) const  
    *Draw an interval of the curve.*
- virtual QwtInterval [boundingInterval](#) (int scaleId) const
- virtual QwtGraphic [legendIcon](#) (int index, const QSizeF &) const

#### Protected Member Functions

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

#### 5.2.1 Detailed Description

An item, that represents a series of points.

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

#### See Also

[QwtPolarPlot](#), [QwtSymbol](#), [QwtScaleMap](#)

#### 5.2.2 Member Enumeration Documentation

##### 5.2.2.1 enum QwtPolarCurve::CurveStyle

Curve styles.

See Also

[setStyle\(\)](#), [style\(\)](#)

Enumerator

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

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

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

### 5.2.2.2 enum QwtPolarCurve::LegendAttribute

Attributes how to represent the curve on the legend.

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

See Also

[setLegendAttribute\(\)](#), [testLegendAttribute\(\)](#)

Enumerator

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

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

## 5.2.3 Constructor & Destructor Documentation

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

Constructor

Parameters

<i>title</i>	title of the curve
--------------	--------------------

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

Constructor

Parameters

<i>title</i>	title of the curve
--------------	--------------------

## 5.2.4 Member Function Documentation

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

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

Parameters

<i>scaleId</i>	Scale index
----------------	-------------

Returns

bounding interval

## See Also

`QwtData::boundingRect()`

Reimplemented from [QwtPolarItem](#).

#### 5.2.4.2 `QwtCurveFitter * QwtPolarCurve::curveFitter ( ) const`

## Returns

The curve fitter

## See Also

[setCurveFitter\(\)](#)

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

## Returns

the the curve data

#### 5.2.4.4 `size_t QwtPolarCurve::dataSize ( ) const`

## Returns

Number of points

## See Also

[setData\(\)](#)

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

Draw the curve

## Parameters

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

Implements [QwtPolarItem](#).

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

Draw an interval of the curve.

## Parameters

<i>painter</i>	Painter
<i>azimuthMap</i>	Maps azimuth values to values related to 0.0, M_2PI

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

See Also

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

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

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

Parameters

<i>painter</i>	Painter
<i>style</i>	Curve style, see <a href="#">QwtPolarCurve::CurveStyle</a>
<i>azimuthMap</i>	Maps azimuth values to values related to 0.0, M_2PI
<i>radialMap</i>	Maps radius values into painter coordinates.
<i>pole</i>	Position of the pole in painter coordinates
<i>from</i>	index of the first point to be painted
<i>to</i>	index of the last point to be painted.

See Also

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

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

Draw lines

Parameters

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

See Also

[draw\(\)](#), [drawLines\(\)](#), [setCurveFitter\(\)](#)

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

Draw symbols

Parameters

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

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

See Also

[setSymbol\(\)](#), [draw\(\)](#), [drawCurve\(\)](#)

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

Returns

Icon representing the curve on the legend

Parameters

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

See Also

[QwtPolarItem::setLegendIconSize\(\)](#), [QwtPolarItem::legendData\(\)](#)

Reimplemented from [QwtPolarItem](#).

5.2.4.11 `const QPen & QwtPolarCurve::pen ( ) const`

Returns

Pen used to draw the lines

See Also

[setPen\(\)](#)

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

Returns

[QwtPolarCurve::Rtti\\_PolarCurve](#)

Reimplemented from [QwtPolarItem](#).

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

Parameters

<i>i</i>	index
----------	-------

Returns

point at position *i*

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

Insert a curve fitter.

## Parameters

<i>curveFitter</i>	Curve fitter
--------------------	--------------

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

## See Also

[curveFitter\(\)](#)

## 5.2.4.15 void QwtPolarCurve::setData ( QwtSeriesData&lt; QwtPointPolar &gt; \* data )

Initialize data with a pointer to QwtSeriesData<QwtPointPolar>.

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

## Parameters

<i>data</i>	Data
-------------	------

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

Specify an attribute how to draw the legend identifier

## Parameters

<i>attribute</i>	Attribute
<i>on</i>	On/Off /sa LegendAttribute, <a href="#">testLegendAttribute()</a>

## 5.2.4.17 void QwtPolarCurve::setPen ( const QPen &amp; pen )

Assign a pen.

## Parameters

<i>pen</i>	New pen
------------	---------

## See Also

[pen\(\)](#)

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

Set the curve's drawing style

## Parameters

<i>style</i>	Curve style
--------------	-------------

## See Also

[CurveStyle](#), [style\(\)](#)

## 5.2.4.19 void QwtPolarCurve::setSymbol ( QwtSymbol \* symbol )

Assign a symbol.

## Parameters

<i>symbol</i>	Symbol
---------------	--------

## See Also

[symbol\(\)](#)

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

## Returns

Current style

## See Also

[CurveStyle](#), [setStyle\(\)](#)

5.2.4.21 **const QwtSymbol \*** QwtPolarCurve::symbol ( ) const

## Returns

The current symbol

## See Also

[setSymbol\(\)](#)

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

Test if a legend attribute is enabled.

## Parameters

<i>attribute</i>	Legend attribute
------------------	------------------

## Returns

True if attribute is enabled

## See Also

[LegendAttribute](#), [setLegendAttribute\(\)](#)

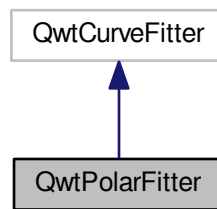
5.3 **QwtPolarFitter Class Reference**

A simple curve fitter for polar points.

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



Inheritance diagram for QwtPolarFitter:



#### Public Member Functions

- [QwtPolarFitter](#) (int [stepCount](#)=5)
- virtual [~QwtPolarFitter](#) ()  
*Destructor.*
- void [setStepCount](#) (int size)
- int [stepCount](#) () const
- virtual QPolygonF [fitCurve](#) (const QPolygonF &) const

#### 5.3.1 Detailed Description

A simple curve fitter for polar points.

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

See Also

[QwtPolarCurve::setCurveFitter\(\)](#)

#### 5.3.2 Constructor & Destructor Documentation

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

Constructor

Parameters

<i>stepCount</i>	Number of points, that will be inserted between 2 points
------------------	----------------------------------------------------------

See Also

[setStepCount\(\)](#)

#### 5.3.3 Member Function Documentation

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

Insert [stepCount\(\)](#) number of additional points between 2 elements of points.

## Parameters

<i>points</i>	Array of points
---------------	-----------------

## Returns

Array of points including the additional points

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

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

## Parameters

<i>stepCount</i>	Number of steps
------------------	-----------------

## See Also

[stepCount\(\)](#)

## 5.3.3.3 int QwtPolarFitter::stepCount ( ) const

## Returns

Number of points, that will be inserted between 2 points

## See Also

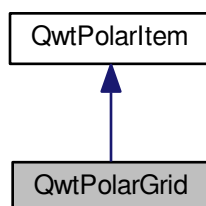
[setStepCount\(\)](#)

## 5.4 QwtPolarGrid Class Reference

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

```
#include <qwt_polar_grid.h>
```

Inheritance diagram for QwtPolarGrid:



## Public Types

- enum [DisplayFlag](#) {  
[SmartOriginLabel](#) = 1, [HideMaxRadiusLabel](#) = 2, [ClipAxisBackground](#) = 4, [SmartScaleDraw](#) = 8,  
[ClipGridLines](#) = 16 }

- enum [GridAttribute](#) { [AutoScaling](#) = 0x01 }
- *Grid attributes.*
- typedef QFlags< [DisplayFlag](#) > [DisplayFlags](#)
- *Display flags.*
- typedef QFlags< [GridAttribute](#) > [GridAttributes](#)
- *Grid attributes.*

#### Public Member Functions

- [QwtPolarGrid](#) ()
- *Constructor.*
- virtual [~QwtPolarGrid](#) ()
- *Destructor.*
- virtual int [rtti](#) () const
- void [setDisplayFlag](#) ([DisplayFlag](#), bool on=true)
- bool [testDisplayFlag](#) ([DisplayFlag](#)) const
- void [setGridAttribute](#) ([GridAttribute](#), bool on=true)
- *Specify an attribute for the grid.*
- bool [testGridAttribute](#) ([GridAttribute](#)) const
- void [showGrid](#) (int scaleId, bool [show](#)=true)
- bool [isGridVisible](#) (int scaleId) const
- void [showMinorGrid](#) (int scaleId, bool [show](#)=true)
- bool [isMinorGridVisible](#) (int scaleId) const
- void [showAxis](#) (int axisId, bool [show](#)=true)
- bool [isAxisVisible](#) (int axisId) const
- void [setPen](#) (const QPen &p)
- void [setFont](#) (const QFont &f)
- void [setMajorGridPen](#) (const QPen &p)
- void [setMajorGridPen](#) (int scaleId, const QPen &p)
- QPen [majorGridPen](#) (int scaleId) const
- void [setMinorGridPen](#) (const QPen &p)
- void [setMinorGridPen](#) (int scaleId, const QPen &p)
- QPen [minorGridPen](#) (int scaleId) const
- void [setAxisPen](#) (int axisId, const QPen &p)
- QPen [axisPen](#) (int axisId) const
- void [setAxisFont](#) (int axisId, const QFont &f)
- QFont [axisFont](#) (int axisId) const
- void [setScaleDraw](#) (int axisId, QwtScaleDraw \*)
- *Set a scale draw.*
- const QwtScaleDraw \* [scaleDraw](#) (int axisId) const
- QwtScaleDraw \* [scaleDraw](#) (int axisId)
- void [setAzimuthScaleDraw](#) (QwtRoundScaleDraw \*)
- *Set a scale draw for the azimuth scale.*
- const QwtRoundScaleDraw \* [azimuthScaleDraw](#) () const
- QwtRoundScaleDraw \* [azimuthScaleDraw](#) ()
- virtual void [draw](#) (QPainter \*p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &rect) const
- virtual void [updateScaleDiv](#) (const QwtScaleDiv &azimuthMap, const QwtScaleDiv &radialMap, const QwtInterval &i)
- *Update the item to changes of the axes scale division.*
- virtual int [marginHint](#) () const

## Protected Member Functions

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

### 5.4.1 Detailed Description

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

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

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

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

See Also

[QwtPolarPlot](#), [QwtPolar::Axis](#)

### 5.4.2 Member Enumeration Documentation

#### 5.4.2.1 enum [QwtPolarGrid::DisplayFlag](#)

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

The default setting enables all flags.

See Also

[setDisplayFlag\(\)](#), [testDisplayFlag\(\)](#)

## Enumerator

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

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

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

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

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

#### 5.4.2.2 enum [QwtPolarGrid::GridAttribute](#)

Grid attributes.

See Also

[setGridAttributes\(\)](#), [testGridAttributes\(\)](#)

## Enumerator

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

## 5.4.3 Constructor &amp; Destructor Documentation

## 5.4.3.1 QwtPolarGrid::QwtPolarGrid ( ) [explicit]

Constructor.

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

## 5.4.4 Member Function Documentation

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

Returns

Font for the tick labels of a specific axis

Parameters

<i>axisId</i>	Axis id (QwtPolar::Axis)
---------------	--------------------------

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

Returns

Pen for painting a specific axis

Parameters

<i>axisId</i>	Axis id (QwtPolar::Axis)
---------------	--------------------------

See Also

[setAxisPen\(\)](#)

## 5.4.4.3 const QwtRoundScaleDraw \* QwtPolarGrid::azimuthScaleDraw ( ) const

Returns

Scale draw for the azimuth scale

See Also

[setAzimuthScaleDraw\(\)](#), [scaleDraw\(\)](#)

## 5.4.4.4 QwtRoundScaleDraw \* QwtPolarGrid::azimuthScaleDraw ( )

Returns

Scale draw for the azimuth scale

See Also

[setAzimuthScaleDraw\(\)](#), [scaleDraw\(\)](#)

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

Draw the grid and axes

## Parameters

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

Implements [QwtPolarItem](#).

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

Paint an axis

## Parameters

<i>painter</i>	Painter
<i>axisId</i>	Axis id (QwtPolar::Axis)

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

Draw circles

## Parameters

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

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

Draw lines from the pole

## Parameters

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

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

## Returns

true if the axis is visible

## Parameters

<i>axisId</i>	Axis id (QwtPolar::Axis)
---------------	--------------------------

## See Also

[showAxis\(\)](#)

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

true if grid lines are enabled

**Parameters**

<i>scaleId</i>	Scale id ( <code>QwtPolar::Scale</code> )
----------------	-------------------------------------------

**See Also**

`QwtPolar::Scale`, [showGrid\(\)](#)

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

true if minor grid lines are enabled

**Parameters**

<i>scaleId</i>	Scale id ( <code>QwtPolar::Scale</code> )
----------------	-------------------------------------------

**See Also**

[showMinorGrid\(\)](#)

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

Pen for painting the major grid lines of a specific scale

**Parameters**

<i>scaleId</i>	Scale id ( <code>QwtPolar::Scale</code> )
----------------	-------------------------------------------

**See Also**

[setMajorGridPen\(\)](#), [minorGridPen\(\)](#)

**5.4.4.13** `int QwtPolarGrid::marginHint ( ) const` `[virtual]`**Returns**

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

**See Also**

`QwtRoundScaleDraw::extent()`

Reimplemented from [QwtPolarItem](#).

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

Pen for painting the minor grid lines of a specific scale

## Parameters

<i>scaleId</i>	Scale id ( QwtPolar::Scale )
----------------	------------------------------

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

## Returns

QwtPlotItem::Rtti\_PolarGrid

Reimplemented from [QwtPolarItem](#).

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

Returns the scale draw of a specified axis

## Parameters

<i>axisId</i>	axis index ( QwtPolar::AxisLeft <= axisId <= QwtPolar::AxisBottom)
---------------	--------------------------------------------------------------------

## Returns

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

## See Also

[azimuthScaleDraw\(\)](#)

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

Returns the scale draw of a specified axis

## Parameters

<i>axisId</i>	axis index ( QwtPolar::AxisLeft <= axisId <= QwtPolar::AxisBottom)
---------------	--------------------------------------------------------------------

## Returns

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

## See Also

[setScaleDraw\(\)](#), [azimuthScaleDraw\(\)](#)

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

Assign a font for the tick labels of a specific axis

## Parameters

<i>axisId</i>	Axis id (QwtPolar::Axis)
<i>font</i>	new Font

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

Assign a pen for painting an axis



## Parameters

<i>axisId</i>	Axis id (QwtPolar::Axis)
<i>pen</i>	Pen

## See Also

[axisPen\(\)](#)

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

Set a scale draw for the azimuth scale.

## Parameters

<i>scaleDraw</i>	object responsible for drawing scales.
------------------	----------------------------------------

## See Also

[azimuthScaleDraw\(\)](#), [setScaleDraw\(\)](#)

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

Change the display flags

## Parameters

<i>flag</i>	See DisplayFlag
<i>on</i>	true/false

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

Assign a font for all scale tick labels

## Parameters

<i>font</i>	Font
-------------	------

## See Also

[setAxisFont\(\)](#)

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

Specify an attribute for the grid.

## Parameters

<i>attribute</i>	Grid attribute
<i>on</i>	On/Off

/sa GridAttribute, [testGridAttribute\(\)](#), [updateScaleDiv\(\)](#), [QwtPolarPlot::zoom\(\)](#), [QwtPolarPlot::scaleDiv\(\)](#)

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

Assign a pen for the major grid lines

## Parameters

<i>pen</i>	Pen
------------	-----

## See Also

[setPen\(\)](#), [setMinorGridPen\(\)](#), [majorGridPen](#)

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

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

## Parameters

<i>scaleId</i>	Scale id ( QwtPolar::Scale )
<i>pen</i>	Pen

## See Also

[setPen\(\)](#), [setMinorGridPen\(\)](#), [majorGridPen](#)

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

Assign a pen for the minor grid lines

## Parameters

<i>pen</i>	Pen
------------	-----

## See Also

[setPen\(\)](#), [setMajorGridPen\(\)](#), [minorGridPen\(\)](#)

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

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

## Parameters

<i>scaleId</i>	Scale id ( QwtPolar::Scale )
<i>pen</i>	Pen

## See Also

[setPen\(\)](#), [setMajorGridPen\(\)](#), [minorGridPen](#)

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

Assign a pen for all axes and grid lines

## Parameters

<i>pen</i>	Pen
------------	-----

## See Also

[setMajorGridPen\(\)](#), [setMinorGridPen\(\)](#), [setAxisPen\(\)](#)

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

Set a scale draw.

## Parameters

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

## See Also

[scaleDraw\(\)](#), [setAzimuthScaleDraw\(\)](#)

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

Show/Hide an axis

## Parameters

<i>axisId</i>	Axis id (QwtPolar::Axis)
<i>show</i>	true/false

## See Also

[isAxisVisible\(\)](#)

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

Show/Hide grid lines for a scale

## Parameters

<i>scaleId</i>	Scale id ( QwtPolar::Scale )
<i>show</i>	true/false

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

Show/Hide minor grid lines for a scale

To display minor grid lines. [showGrid\(\)](#) needs to be enabled too.

## Parameters

<i>scaleId</i>	Scale id ( QwtPolar::Scale )
<i>show</i>	true/false

## See Also

[showGrid](#)

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

## Returns

true, if flag is enabled

## Parameters

<i>flag</i>	See DisplayFlag
-------------	-----------------

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

## Returns

true, if attribute is enabled

See Also

[GridAttribute](#), [setGridAttribute\(\)](#)

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

Update the item to changes of the axes scale division.

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

Parameters

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

See Also

[QwtPolarPlot::setGridAttributes\(\)](#)

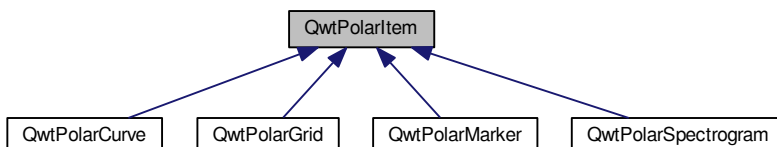
Reimplemented from [QwtPolarItem](#).

## 5.5 QwtPolarItem Class Reference

Base class for items on a polar plot.

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

Inheritance diagram for QwtPolarItem:



Public Types

- enum [RttiValues](#) { [Rtti\\_PolarItem](#) = 0, [Rtti\\_PolarGrid](#), [Rtti\\_PolarMarker](#), [Rtti\\_PolarCurve](#), [Rtti\\_PolarSpectrogram](#), [Rtti\\_PolarUserItem](#) = 1000 }

*Runtime type information.*

- enum [ItemAttribute](#) { [Legend](#) = 0x01, [AutoScale](#) = 0x02 }

*Plot Item Attributes.*

- enum [RenderHint](#) { [RenderAntialiased](#) = 0x01 }

*Render hints.*

- typedef QFlags< [ItemAttribute](#) > [ItemAttributes](#)

*Item attributes.*

- typedef QFlags< [RenderHint](#) > [RenderHints](#)

*Item attributes.*

## Public Member Functions

- [QwtPolarItem](#) (const QwtText &title=QwtText())
- virtual [~QwtPolarItem](#) ()  
*Destroy the [QwtPolarItem](#).*
- void [attach](#) ([QwtPolarPlot](#) \*plot)  
*Attach the item to a plot.*
- void [detach](#) ()  
*This method detaches a [QwtPolarItem](#) from the [QwtPolarPlot](#) it has been associated with.*
- [QwtPolarPlot](#) \* [plot](#) () const
- void [setTitle](#) (const QString &title)
- void [setTitle](#) (const QwtText &title)
- const QwtText & [title](#) () const
- virtual int [rtti](#) () const
- void [setItemAttribute](#) (ItemAttribute, bool on=true)
- bool [testItemAttribute](#) (ItemAttribute) const
- void [setRenderHint](#) (RenderHint, bool on=true)
- bool [testRenderHint](#) (RenderHint) const
- void [setRenderThreadCount](#) (uint numThreads)
- uint [renderThreadCount](#) () const
- double [z](#) () const
- void [setZ](#) (double z)  
*Set the z value.*
- void [show](#) ()  
*Show the item.*
- void [hide](#) ()  
*Hide the item.*
- virtual void [setVisible](#) (bool)
- bool [isVisible](#) () const
- virtual void [itemChanged](#) ()
- virtual void [legendChanged](#) ()
- virtual void [draw](#) (QPainter \*painter, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const =0  
*Draw the item.*
- virtual QwtInterval [boundingInterval](#) (int scaleId) const
- virtual void [updateScaleDiv](#) (const QwtScaleDiv &, const QwtScaleDiv &, const QwtInterval &)  
*Update the item to changes of the axes scale division.*
- virtual int [marginHint](#) () const
- void [setLegendIconSize](#) (const QSize &)
- QSize [legendIconSize](#) () const
- virtual QList< QwtLegendData > [legendData](#) () const  
*Return all information, that is needed to represent the item on the legend.*
- virtual QwtGraphic [legendIcon](#) (int index, const QSizeF &) const

## 5.5.1 Detailed Description

Base class for items on a polar plot.

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

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

## 5.5.2 Member Enumeration Documentation

### 5.5.2.1 enum `QwtPolarItem::ItemAttribute`

Plot Item Attributes.

See Also

[setItemAttribute\(\)](#), [testItemAttribute\(\)](#)

Enumerator

**Legend** The item is represented on the legend.

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

### 5.5.2.2 enum `QwtPolarItem::RenderHint`

Render hints.

See Also

[setRenderHint\(\)](#), [testRenderHint\(\)](#)

Enumerator

**RenderAntialiased** Enable antialiasing.

### 5.5.2.3 enum `QwtPolarItem::RttiValues`

Runtime type information.

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

Enumerator

**Rtti\_PolarItem** Unspecific value, that can be used, when it doesn't matter.

**Rtti\_PolarGrid** For [QwtPolarGrid](#).

**Rtti\_PolarMarker** For [QwtPolarMarker](#).

**Rtti\_PolarCurve** For [QwtPolarCurve](#).

**Rtti\_PolarSpectrogram** For [QwtPolarSpectrogram](#).

**Rtti\_PolarUserItem** Values  $\geq$  `Rtti_PolarUserItem` are reserved for plot items not implemented in the Qwt-Polar library.

## 5.5.3 Constructor & Destructor Documentation

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

Constructor

Parameters

<i>title</i>	Item title, f.e used on a legend
--------------	----------------------------------

See Also

[setTitle\(\)](#)

## 5.5.4 Member Function Documentation

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

Attach the item to a plot.

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

## Parameters

<i>plot</i>	Plot widget
-------------	-------------

## See Also

[QwtPolarItem::detach\(\)](#)

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

Interval, that is necessary to display the item

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

## Parameters

<i>scaleId</i>	Scale id ( QwtPolar::Scale )
----------------	------------------------------

## Returns

Bounding interval of the plot item for a specific scale

Reimplemented in [QwtPolarCurve](#), [QwtPolarSpectrogram](#), and [QwtPolarMarker](#).

## 5.5.4.3 void QwtPolarItem::detach ( )

This method detaches a [QwtPolarItem](#) from the [QwtPolarPlot](#) it has been associated with.

[detach\(\)](#) is equivalent to calling `attach( NULL )`

## See Also

[attach\(\)](#)

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

Draw the item.

## Parameters

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

<i>canvasRect</i>	Contents rect of the canvas in painter coordinates
-------------------	----------------------------------------------------

Implemented in [QwtPolarGrid](#), [QwtPolarCurve](#), [QwtPolarSpectrogram](#), and [QwtPolarMarker](#).

#### 5.5.4.5 `bool QwtPolarItem::isVisible ( ) const`

Returns

true if visible

See Also

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

#### 5.5.4.6 `void QwtPolarItem::itemChanged ( ) [virtual]`

Update the legend and call [QwtPolarPlot::autoRefresh](#) for the parent plot.

See Also

[updateLegend\(\)](#)

#### 5.5.4.7 `void QwtPolarItem::legendChanged ( ) [virtual]`

Update the legend of the parent plot.

See Also

[QwtPolarPlot::updateLegend\(\)](#), [itemChanged\(\)](#)

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

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

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

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

The default implementation returns one entry with the [title\(\)](#) of the item and the [legendIcon\(\)](#).

See Also

[title\(\)](#), [legendIcon\(\)](#), [QwtLegend](#)

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

Returns

Icon representing the item on the legend

The default implementation returns an invalid icon

Parameters

<i>index</i>	Index of the legend entry ( usually there is only one )
<i>size</i>	Icon size

See Also

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

Reimplemented in [QwtPolarCurve](#).



#### 5.5.4.10 QSize QwtPolarItem::legendIconSize ( ) const

##### Returns

Legend icon size

##### See Also

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

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

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

##### Returns

Hint for the margin

Reimplemented in [QwtPolarGrid](#).

#### 5.5.4.12 QwtPolarPlot \* QwtPolarItem::plot ( ) const

##### Returns

Attached plot

#### 5.5.4.13 uint QwtPolarItem::renderThreadCount ( ) const

##### Returns

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

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

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

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

##### Returns

rtti value

##### See Also

[RttiValues](#)

Reimplemented in [QwtPolarGrid](#), [QwtPolarCurve](#), [QwtPolarSpectrogram](#), and [QwtPolarMarker](#).

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

Toggle an item attribute

## Parameters

<i>attribute</i>	Attribute type
<i>on</i>	true/false

## See Also

[testItemAttribute\(\)](#), [ItemAttribute](#)

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

Set the size of the legend icon

The default setting is 8x8 pixels

## Parameters

<i>size</i>	Size
-------------	------

## See Also

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

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

Toggle an render hint

## Parameters

<i>hint</i>	Render hint
<i>on</i>	true/false

## See Also

[testRenderHint\(\)](#), [RenderHint](#)

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

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

The default setting is set to 1.

## Parameters

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

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

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

Set a new title

## Parameters

<i>title</i>	Title
--------------	-------

## See Also

[title\(\)](#)

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

Set a new title

## Parameters

<i>title</i>	Title
--------------	-------

## See Also

[title\(\)](#)

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

Show/Hide the item

## Parameters

<i>on</i>	Show if true, otherwise hide
-----------	------------------------------

## See Also

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

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

Set the z value.

Plot items are painted in increasing z-order.

## Parameters

<i>z</i>	Z-value
----------	---------

## See Also

[z\(\)](#), [QwtPolarItemDict::itemList\(\)](#)

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

Test an item attribute

## Parameters

<i>attribute</i>	Attribute type
------------------	----------------

## Returns

true/false

## See Also

[setItemAttribute\(\)](#), [ItemAttribute](#)

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

Test a render hint

## Parameters

---

<i>hint</i>	Render hint
-------------	-------------

**Returns**

true/false

**See Also**[setRenderHint\(\)](#), [RenderHint](#)**5.5.4.25 const QwtText & QwtPolarItem::title ( ) const****Returns**

Title of the item

**See Also**[setTitle\(\)](#)**5.5.4.26 void QwtPolarItem::updateScaleDiv ( const QwtScaleDiv & azimuthScaleDiv, const QwtScaleDiv & radialScaleDiv, const QwtInterval & interval ) [virtual]**

Update the item to changes of the axes scale division.

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

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

**See Also**[QwtPolarPlot::updateAxes\(\)](#)Reimplemented in [QwtPolarGrid](#).**5.5.4.27 double QwtPolarItem::z ( ) const**

Plot items are painted in increasing z-order.

**Returns**

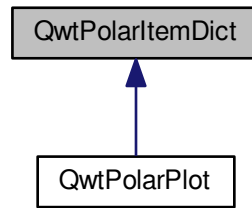
Z value

**See Also**[setZ\(\)](#), [QwtPolarItemDict::itemList\(\)](#)**5.6 QwtPolarItemDict Class Reference**

A dictionary for polar plot items.

#include &lt;qwt\_polar\_itemdict.h&gt;

Inheritance diagram for QwtPolarItemDict:



#### Public Member Functions

- [QwtPolarItemDict](#) ()
- [~QwtPolarItemDict](#) ()
- void [setAutoDelete](#) (bool)
- bool [autoDelete](#) () const
- const QwtPolarItemList & [itemList](#) () const  
*A QwtPolarItemList of all attached plot items.*
- void [detachItems](#) (int rtti=[QwtPolarItem::Rtti\\_PolarItem](#), bool [autoDelete](#)=true)

#### Protected Member Functions

- void [insertItem](#) ([QwtPolarItem](#) \*)
- void [removeItem](#) ([QwtPolarItem](#) \*)

#### 5.6.1 Detailed Description

A dictionary for polar plot items.

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

#### See Also

[QwtPolarItem::attach\(\)](#), [QwtPolarItem::detach\(\)](#), [QwtPolarItem::z\(\)](#)

#### 5.6.2 Constructor & Destructor Documentation

##### 5.6.2.1 QwtPolarItemDict::QwtPolarItemDict ( ) [explicit]

Constructor

Auto deletion is enabled.

#### See Also

[setAutoDelete](#), [attachItem](#)

### 5.6.2.2 QwtPolarItemDict::~~QwtPolarItemDict ( )

Destructor

If autoDelete is on, all attached items will be deleted

See Also

[setAutoDelete](#), [autoDelete](#), [attachItem](#)

## 5.6.3 Member Function Documentation

### 5.6.3.1 bool QwtPolarItemDict::autoDelete ( ) const

Returns

true if auto deletion is enabled

See Also

[setAutoDelete](#), [attachItem](#)

### 5.6.3.2 void QwtPolarItemDict::detachItems ( int *rtti* = QwtPolarItem::Rtti\_PolarItem, bool *autoDelete* = true )

Detach items from the dictionary

Parameters

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

### 5.6.3.3 void QwtPolarItemDict::insertItem ( QwtPolarItem \* *item* ) [protected]

Insert a plot item

Parameters

<i>item</i>	PlotItem
-------------	----------

See Also

[removeItem\(\)](#)

### 5.6.3.4 const QwtPolarItemList & QwtPolarItemDict::itemList ( ) const

A QwtPolarItemList of all attached plot items.

Returns

List of all attached plot items.

Note

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

### 5.6.3.5 void QwtPolarItemDict::removeItem ( QwtPolarItem \* *item* ) [protected]

Remove a plot item

## Parameters

<i>item</i>	PlotItem
-------------	----------

## See Also

[insertItem\(\)](#)

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

En/Disable Auto deletion

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

## See Also

[autoDelete](#), [attachItem](#)

## 5.7 QwtPolarLayout Class Reference

Layout class for [QwtPolarPlot](#).

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

## Public Types

- enum [Option](#) { [IgnoreScrollbars](#) = 0x01, [IgnoreFrames](#) = 0x02, [IgnoreTitle](#) = 0x04, [IgnoreLegend](#) = 0x08 }
- *Options to configure the plot layout engine.*
- typedef QFlags< [Option](#) > [Options](#)
- *Options to configure the plot layout engine.*

## Public Member Functions

- [QwtPolarLayout](#) ()
- *Constructor.*
- virtual [~QwtPolarLayout](#) ()
- *Destructor.*
- void [setLegendPosition](#) ([QwtPolarPlot::LegendPosition](#) pos, double ratio)
- *Specify the position of the legend.*
- void [setLegendPosition](#) ([QwtPolarPlot::LegendPosition](#) pos)
- *Specify the position of the legend.*
- [QwtPolarPlot::LegendPosition legendPosition](#) () const
- void [setLegendRatio](#) (double ratio)
- double [legendRatio](#) () const
- virtual void [activate](#) (const [QwtPolarPlot](#) \*, const QRectF &rect, [Options](#) options=0)
- *Recalculate the geometry of all components.*
- virtual void [invalidate](#) ()
- const QRectF & [titleRect](#) () const
- const QRectF & [legendRect](#) () const
- const QRectF & [canvasRect](#) () const

## Protected Member Functions

- QRectF [layoutLegend](#) ([Options](#) options, QRectF &) const

### 5.7.1 Detailed Description

Layout class for [QwtPolarPlot](#).

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

### 5.7.2 Member Enumeration Documentation

#### 5.7.2.1 enum QwtPolarLayout::Option

Options to configure the plot layout engine.

Enumerator

**IgnoreScrollbars** Ignore the dimension of the scrollbars.

**IgnoreFrames** Ignore all frames.

**IgnoreTitle** Ignore the title.

**IgnoreLegend** Ignore the legend.

### 5.7.3 Member Function Documentation

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

Recalculate the geometry of all components.

Parameters

<i>plot</i>	Plot to be layout
<i>boundingRect</i>	Rect where to place the components
<i>options</i>	Options

See Also

[invalidate\(\)](#), [titleRect\(\)](#), [legendRect\(\)](#), [canvasRect\(\)](#)

#### 5.7.3.2 const QRectF & QwtPolarLayout::canvasRect ( ) const

Returns

Geometry for the canvas

See Also

[activate\(\)](#), [invalidate\(\)](#)

#### 5.7.3.3 void QwtPolarLayout::invalidate ( ) [virtual]

Invalidate the geometry of all components.

See Also

[activate\(\)](#)

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

Find the geometry for the legend



## Parameters

<i>options</i>	Options how to layout the legend
<i>rect</i>	Rectangle where to place the legend

## Returns

Geometry for the legend

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

## Returns

Position of the legend

## See Also

[setLegendPosition\(\)](#), [QwtPolarPlot::setLegendPosition\(\)](#), [QwtPolarPlot::legendPosition\(\)](#)

## 5.7.3.6 double QwtPolarLayout::legendRatio ( ) const

## Returns

The relative size of the legend in the plot.

## See Also

[setLegendPosition\(\)](#)

## 5.7.3.7 const QRectF &amp; QwtPolarLayout::legendRect ( ) const

## Returns

Geometry for the legend

## See Also

[activate\(\)](#), [invalidate\(\)](#)

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

Specify the position of the legend.

## Parameters

<i>pos</i>	The legend's position.
<i>ratio</i>	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to ]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

## See Also

[QwtPolarPlot::setLegendPosition\(\)](#)

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

Specify the position of the legend.

## Parameters

<i>pos</i>	The legend's position. Valid values are <code>QwtPolarPlot::LeftLegend</code> , <code>QwtPolarPlot::RightLegend</code> , <code>QwtPolarPlot::TopLegend</code> , <code>QwtPolarPlot::BottomLegend</code> .
------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## See Also

`QwtPolarPlot::setLegendPosition()`

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

Specify the relative size of the legend in the plot

## Parameters

<i>ratio</i>	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to ]0.0 .. 1.0]. In case of $\leq 0.0$ it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.
--------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5.7.3.11 `const QRectF & QwtPolarLayout::titleRect ( ) const`

## Returns

Geometry for the title

## See Also

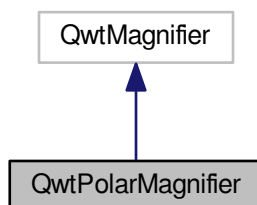
`activate()`, `invalidate()`

## 5.8 QwtPolarMagnifier Class Reference

[QwtPolarMagnifier](#) provides zooming, by magnifying in steps.

```
#include <qwt_polar_magnifier.h>
```

Inheritance diagram for QwtPolarMagnifier:



## Public Member Functions

- [QwtPolarMagnifier](#) ([QwtPolarCanvas](#) \*)
- virtual `~QwtPolarMagnifier` ()

*Destructor.*

- void [setUnzoomKey](#) (int key, int modifiers)
- void [getUnzoomKey](#) (int &key, int &modifiers) const
- [QwtPolarPlot](#) \* [plot](#) ()
- const [QwtPolarPlot](#) \* [plot](#) () const
- [QwtPolarCanvas](#) \* [canvas](#) ()
- const [QwtPolarCanvas](#) \* [canvas](#) () const

#### Protected Member Functions

- virtual void [rescale](#) (double factor)
- void [unzoom](#) ()

*Unzoom the plot widget.*

- virtual void [widgetKeyPressEvent](#) (QKeyEvent \*)

#### 5.8.1 Detailed Description

[QwtPolarMagnifier](#) provides zooming, by magnifying in steps.

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

Together with [QwtPolarPanner](#) it is possible to implement an individual navigation of the plot canvas.

See Also

[QwtPolarPanner](#), [QwtPolarPlot](#), [QwtPolarCanvas](#)

#### 5.8.2 Constructor & Destructor Documentation

##### 5.8.2.1 [QwtPolarMagnifier::QwtPolarMagnifier](#) ( [QwtPolarCanvas](#) \* *canvas* ) `[explicit]`

Constructor

Parameters

<i>canvas</i>	Plot canvas to be magnified
---------------	-----------------------------

#### 5.8.3 Member Function Documentation

##### 5.8.3.1 [QwtPolarCanvas](#) \* [QwtPolarMagnifier::canvas](#) ( )

Returns

Observed plot canvas

##### 5.8.3.2 const [QwtPolarCanvas](#) \* [QwtPolarMagnifier::canvas](#) ( ) const

Returns

Observed plot canvas

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

Returns

Key, and modifiers that are used for unzooming

## Parameters

<i>key</i>	Key code
<i>modifiers</i>	Modifiers

## See Also

[setUnzoomKey\(\)](#), [QwtPolarPlot::unzoom\(\)](#)

5.8.3.4 **QwtPolarPlot \* QwtPolarMagnifier::plot ( )**

## Returns

Observed plot

5.8.3.5 **const QwtPolarPlot \* QwtPolarMagnifier::plot ( ) const**

## Returns

observed plot

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

Zoom in/out the zoomed area

## Parameters

<i>factor</i>	A value < 1.0 zooms in, a value > 1.0 zooms out.
---------------	--------------------------------------------------

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

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

## Parameters

<i>key</i>	Key code
<i>modifiers</i>	Modifiers

## See Also

[getUnzoomKey\(\)](#), [QwtPolarPlot::unzoom\(\)](#)

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

Handle a key press event for the observed widget.

## Parameters

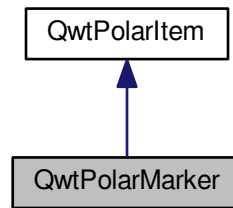
<i>event</i>	Key event
--------------	-----------

5.9 **QwtPolarMarker Class Reference**

A class for drawing markers.

```
#include <qwt_polar_marker.h>
```

Inheritance diagram for QwtPolarMarker:



### Public Member Functions

- [QwtPolarMarker](#) ()  
*Sets alignment to Qt::AlignCenter, and style to NoLine.*
- virtual [~QwtPolarMarker](#) ()  
*Destructor.*
- virtual int [rtti](#) () const
- void [setPosition](#) (const QwtPointPolar &)  
*Change the position of the marker.*
- QwtPointPolar [position](#) () const
- void [setSymbol](#) (const QwtSymbol \*s)  
*Assign a symbol.*
- const QwtSymbol \* [symbol](#) () const
- void [setLabel](#) (const QwtText &)  
*Set the label.*
- QwtText [label](#) () const
- void [setLabelAlignment](#) (Qt::Alignment)  
*Set the alignment of the label.*
- Qt::Alignment [labelAlignment](#) () const
- virtual void [draw](#) (QPainter \*painter, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const
- virtual QwtInterval [boundingInterval](#) (int scaleId) const

### Additional Inherited Members

#### 5.9.1 Detailed Description

A class for drawing markers.

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

The [setSymbol\(\)](#) member assigns a symbol to the marker. The symbol is drawn at the specified point.

With [setLabel\(\)](#), a label can be assigned to the marker. The [setLabelAlignment\(\)](#) member specifies where the label is drawn. All the Align\*-constants in Qt::AlignmentFlags (see Qt documentation) are valid. The alignment refers to the center point of the marker, which means, for example, that the label would be painted left above the center point if the alignment was set to AlignLeft|AlignTop.

## 5.9.2 Member Function Documentation

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

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

Parameters

<i>scaleId</i>	Scale index
----------------	-------------

Returns

bounding interval ( == position )

See Also

[position\(\)](#)

Reimplemented from [QwtPolarItem](#).

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

Draw the marker

Parameters

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

Implements [QwtPolarItem](#).

### 5.9.2.3 `QwtText QwtPolarMarker::label ( ) const`

Returns

the label

See Also

[setLabel\(\)](#)

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

Returns

the label alignment

See Also

[setLabelAlignment\(\)](#)

### 5.9.2.5 `QwtPointPolar QwtPolarMarker::position ( ) const`

Returns

Position of the marker

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

Returns

`QwtPolarItem::Rtti_PlotMarker`

Reimplemented from [QwtPolarItem](#).

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

Set the label.

Parameters

<i>label</i>	label text
--------------	------------

See Also

[label\(\)](#)

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

Set the alignment of the label.

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

Parameters

<i>align</i>	Alignment. A combination of AlignTop, AlignBottom, AlignLeft, AlignRight, AlignCenter, AlignHCenter, AlignVCenter.
--------------	--------------------------------------------------------------------------------------------------------------------

See Also

[labelAlignment\(\)](#)

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

Assign a symbol.

Parameters

<i>symbol</i>	New symbol
---------------	------------

See Also

[symbol\(\)](#)

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

Returns

the symbol

See Also

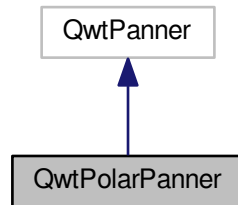
[setSymbol\(\)](#), [QwtSymbol](#)

## 5.10 QwtPolarPanner Class Reference

[QwtPolarPanner](#) provides panning of a polar plot canvas.

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

Inheritance diagram for QwtPolarPanner:



#### Public Member Functions

- [QwtPolarPanner](#) ([QwtPolarCanvas](#) \*)  
*Create a plot panner for a polar plot canvas.*
- virtual [~QwtPolarPanner](#) ()  
*Destructor.*
- [QwtPolarPlot](#) \* [plot](#) ()
- const [QwtPolarPlot](#) \* [plot](#) () const
- [QwtPolarCanvas](#) \* [canvas](#) ()
- const [QwtPolarCanvas](#) \* [canvas](#) () const

#### Protected Slots

- virtual void [movePlot](#) (int dx, int dy)

#### Protected Member Functions

- virtual void [widgetMouseEvent](#) (QMouseEvent \*)

#### 5.10.1 Detailed Description

[QwtPolarPanner](#) provides panning of a polar plot canvas.

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

Together with [QwtPolarMagnifier](#) individual ways of navigating on a [QwtPolarPlot](#) widget can be implemented easily.

#### See Also

[QwtPolarMagnifier](#)

#### 5.10.2 Member Function Documentation

##### 5.10.2.1 [QwtPolarCanvas](#) \* [QwtPolarPanner::canvas](#) ( )



## Returns

observed plot canvas

5.10.2.2 `const QwtPolarCanvas * QwtPolarPanner::canvas ( ) const`

## Returns

observed plot canvas

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

Adjust the zoomed area according to dx/dy

## Parameters

<i>dx</i>	Pixel offset in x direction
<i>dy</i>	Pixel offset in y direction

## See Also

`QwtPanner::panned()`, [QwtPolarPlot::zoom\(\)](#)

5.10.2.4 `QwtPolarPlot * QwtPolarPanner::plot ( )`

## Returns

observed plot

5.10.2.5 `const QwtPolarPlot * QwtPolarPanner::plot ( ) const`

## Returns

observed plot

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

Block panning when the plot zoom factor is  $\geq 1.0$ .

## Parameters

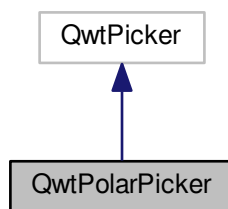
<i>event</i>	Mouse event
--------------	-------------

## 5.11 QwtPolarPicker Class Reference

[QwtPolarPicker](#) provides selections on a plot canvas.

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

Inheritance diagram for QwtPolarPicker:



### Signals

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

### Public Member Functions

- [QwtPolarPicker](#) ([QwtPolarCanvas](#) \*)  
*Create a polar plot picker.*
- virtual [~QwtPolarPicker](#) ()  
*Destructor.*
- [QwtPolarPicker](#) (RubberBand rubberBand, DisplayMode trackerMode, [QwtPolarCanvas](#) \*)
- [QwtPolarPlot](#) \* [plot](#) ()
- const [QwtPolarPlot](#) \* [plot](#) () const
- [QwtPolarCanvas](#) \* [canvas](#) ()
- const [QwtPolarCanvas](#) \* [canvas](#) () const
- virtual QRect [pickRect](#) () const

### Protected Member Functions

- QwtPointPolar [invTransform](#) (const QPoint &) const
- virtual QwtText [trackerText](#) (const QPoint &) const
- virtual QwtText [trackerTextPolar](#) (const QwtPointPolar &) const  
*Translate a position into a position string.*
- virtual void [move](#) (const QPoint &)
- virtual void [append](#) (const QPoint &)
- virtual bool [end](#) (bool ok=true)

#### 5.11.1 Detailed Description

[QwtPolarPicker](#) provides selections on a plot canvas.

[QwtPolarPicker](#) is a [QwtPicker](#) tailored for selections on a polar plot canvas.

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 QwtPolarPicker::QwtPolarPicker ( QwtPolarCanvas \* *canvas* ) [explicit]

Create a polar plot picker.

## Parameters

<i>canvas</i>	Plot canvas to observe, also the parent object
---------------	------------------------------------------------

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

Create a plot picker

## Parameters

<i>rubberBand</i>	Rubberband style
<i>trackerMode</i>	Tracker mode
<i>canvas</i>	Plot canvas to observe, also the parent object

## See Also

QwtPicker, QwtPicker::setSelectionFlags(), QwtPicker::setRubberBand(), QwtPicker::setTrackerMode  
[QwtPolarPlot::autoReplot\(\)](#), [QwtPolarPlot::replot\(\)](#), [scaleRect\(\)](#)

## 5.11.3 Member Function Documentation

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

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

## Parameters

<i>pos</i>	Additional point
------------	------------------

## See Also

isActive, begin(), [end\(\)](#), [move\(\)](#), [appended\(\)](#)

## Note

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

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

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

## Parameters

<i>pos</i>	Position of the appended point.
------------	---------------------------------

## See Also

[append\(\). moved\(\)](#)

5.11.3.3 **QwtPolarCanvas \* QwtPolarPicker::canvas** ( )

## Returns

Observed plot canvas

5.11.3.4 **const QwtPolarCanvas \* QwtPolarPicker::canvas** ( ) const

## Returns

Observed plot canvas

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

Close a selection setting the state to inactive.

**Parameters**

<i>ok</i>	If true, complete the selection and emit selected signals otherwise discard the selection.
-----------	--------------------------------------------------------------------------------------------

**Returns**

true if the selection is accepted, false otherwise

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

Translate a point from widget into plot coordinates

**Parameters**

<i>pos</i>	Point in widget coordinates of the plot canvas
------------	------------------------------------------------

**Returns**

Point in plot coordinates

**See Also**

`transform()`, [canvas\(\)](#)

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

Move the last point of the selection

**Parameters**

<i>pos</i>	New position
------------	--------------

**See Also**

`isActive`, `begin()`, [end\(\)](#), [append\(\)](#)

**Note**

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

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

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

**Parameters**

<i>pos</i>	Position of the moved last point of the selection.
------------	----------------------------------------------------

**See Also**

[move\(\)](#), [appended\(\)](#)

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

**Returns**

Bounding rectangle of the region, where picking is supported.

## 5.11.3.10 QwtPolarPlot \* QwtPolarPicker::plot ( )

## Returns

Plot widget, containing the observed plot canvas

## 5.11.3.11 const QwtPolarPlot \* QwtPolarPicker::plot ( ) const

## Returns

Plot widget, containing the observed plot canvas

## 5.11.3.12 void QwtPolarPicker::selected ( const QwtPointPolar &amp; pos ) [signal]

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

## Parameters

<i>pos</i>	Selected point
------------	----------------

## 5.11.3.13 void QwtPolarPicker::selected ( const QVector&lt; QwtPointPolar &gt; &amp; points ) [signal]

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

## Parameters

<i>points</i>	Selected points
---------------	-----------------

## 5.11.3.14 QwtText QwtPolarPicker::trackerText ( const QPoint &amp; pos ) const [protected], [virtual]

Translate a pixel position into a position string

## Parameters

<i>pos</i>	Position in pixel coordinates
------------	-------------------------------

## Returns

Position string

## 5.11.3.15 QwtText QwtPolarPicker::trackerTextPolar ( const QwtPointPolar &amp; pos ) const [protected], [virtual]

Translate a position into a position string.

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

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

## Parameters

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

## Returns

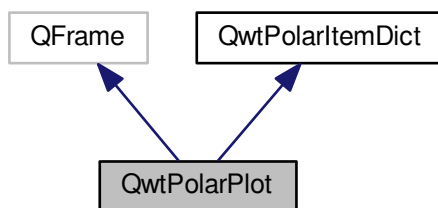
Position string

## 5.12 QwtPolarPlot Class Reference

A plotting widget, displaying a polar coordinate system.

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

Inheritance diagram for QwtPolarPlot:



### Public Types

- enum [LegendPosition](#) {  
[LeftLegend](#), [RightLegend](#), [BottomLegend](#), [TopLegend](#),  
[ExternalLegend](#) }

### Public Slots

- virtual void [replot](#) ()  
*Redraw the plot.*
- void [autoRefresh](#) ()  
*Replots the plot if QwtPlot::autoReplot() is true.*
- void [setAzimuthOrigin](#) (double)  
*Change the origin of the azimuth scale.*

### Signals

- void [itemAttached](#) ([QwtPolarItem](#) \*plotItem, bool on)
- void [legendDataChanged](#) (const QVariant &itemInfo, const QList< [QwtLegendData](#) > &data)
- void [layoutChanged](#) ()

### Public Member Functions

- [QwtPolarPlot](#) (QWidget \*parent=NULL)
- [QwtPolarPlot](#) (const QwtText &title, QWidget \*parent=NULL)
- virtual [~QwtPolarPlot](#) ()  
*Destructor.*
- void [setTitle](#) (const QString &)
- void [setTitle](#) (const QwtText &)
- QwtText [title](#) () const
- QwtTextLabel \* [titleLabel](#) ()
- const QwtTextLabel \* [titleLabel](#) () const
- void [setAutoReplot](#) (bool tf=true)  
*Set or reset the autoReplot option.*
- bool [autoReplot](#) () const
- void [setAutoScale](#) (int scaled)



*Enable autoscaling.*

- bool [hasAutoScale](#) (int scaleId) const
- void [setScaleMaxMinor](#) (int scaleId, int maxMinor)
- int [scaleMaxMinor](#) (int scaleId) const
- int [scaleMaxMajor](#) (int scaleId) const
- void [setScaleMaxMajor](#) (int scaleId, int maxMajor)
- QwtScaleEngine \* [scaleEngine](#) (int scaleId)
- const QwtScaleEngine \* [scaleEngine](#) (int scaleId) const
- void [setScaleEngine](#) (int scaleId, QwtScaleEngine \*)
- void [setScale](#) (int scaleId, double min, double max, double step=0)

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

- void [setScaleDiv](#) (int scaleId, const QwtScaleDiv &)

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

- const QwtScaleDiv \* [scaleDiv](#) (int scaleId) const

*Return the scale division of a specified scale.*

- QwtScaleDiv \* [scaleDiv](#) (int scaleId)

*Return the scale division of a specified scale.*

- QwtScaleMap [scaleMap](#) (int scaleId, double radius) const
- QwtScaleMap [scaleMap](#) (int scaleId) const
- void [updateScale](#) (int scaleId)
- double [azimuthOrigin](#) () const
- void [zoom](#) (const QwtPointPolar &, double factor)

*Translate and in/decrease the zoom factor.*

- void [unzoom](#) ()
- QwtPointPolar [zoomPos](#) () const
- double [zoomFactor](#) () const
- [QwtPolarCanvas](#) \* [canvas](#) ()
- const [QwtPolarCanvas](#) \* [canvas](#) () const
- void [setPlotBackground](#) (const QBrush &c)

*Set the background of the plot area.*

- const QBrush & [plotBackground](#) () const
- virtual void [drawCanvas](#) (QPainter \*, const QRectF &) const
- void [insertLegend](#) (QwtAbstractLegend \*, [LegendPosition=RightLegend](#), double ratio=-1.0)

*Insert a legend.*

- QwtAbstractLegend \* [legend](#) ()
- const QwtAbstractLegend \* [legend](#) () const
- void [updateLegend](#) ()
- void [updateLegend](#) (const [QwtPolarItem](#) \*)
- [QwtPolarLayout](#) \* [plotLayout](#) ()
- const [QwtPolarLayout](#) \* [plotLayout](#) () const
- QwtInterval [visibleInterval](#) () const
- QRectF [plotRect](#) () const
- QRectF [plotRect](#) (const QRectF &) const

*Calculate the bounding rect of the plot area.*

- int [plotMarginHint](#) () const
- virtual QVariant [itemToInfo](#) ([QwtPolarItem](#) \*) const

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

- virtual [QwtPolarItem](#) \* [infoToItem](#) (const QVariant &) const

*Identify the plot item according to an item info object, that has been generated from [itemToInfo\(\)](#).*

## Protected Member Functions

- virtual bool [event](#) (QEvent \*)  
*Qt event handler.*
- virtual void [resizeEvent](#) (QResizeEvent \*)  
*Resize and update internal layout.*
- virtual void [updateLayout](#) ()  
*Rebuild the layout.*
- virtual void [drawItems](#) (QPainter \*painter, const QwtScaleMap &radialMap, const QwtScaleMap &azimuthMap, const QPointF &pole, double radius, const QRectF &canvasRect) const

## Friends

- class [QwtPolarItem](#)

### 5.12.1 Detailed Description

A plotting widget, displaying a polar coordinate system.

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

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

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

### 5.12.2 Member Enumeration Documentation

#### 5.12.2.1 enum [QwtPolarPlot::LegendPosition](#)

Position of the legend, relative to the canvas.

#### See Also

[insertLegend\(\)](#)

#### Enumerator

**LeftLegend** The legend will be left from the canvas.

**RightLegend** The legend will be right from the canvas.

**BottomLegend** The legend will be below the canvas.

**TopLegend** The legend will be between canvas and title.

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

#### Note

The legend is not painted by [QwtPolarRenderer](#)

### 5.12.3 Constructor & Destructor Documentation

#### 5.12.3.1 [QwtPolarPlot::QwtPolarPlot](#) ( QWidget \* *parent* = NULL ) [explicit]

#### Constructor

## Parameters

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

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

## Constructor

## Parameters

<i>title</i>	Title text
<i>parent</i>	Parent widget

## 5.12.4 Member Function Documentation

5.12.4.1 **bool QwtPolarPlot::autoReplot** ( ) const

## Returns

true if the autoReplot option is set.

5.12.4.2 **double QwtPolarPlot::azimuthOrigin** ( ) const

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

## Returns

Origin of the azimuth scale

## See Also

[setAzimuthOrigin\(\)](#)

5.12.4.3 **QwtPolarCanvas \* QwtPolarPlot::canvas** ( )

## Returns

the plot's canvas

5.12.4.4 **const QwtPolarCanvas \* QwtPolarPlot::canvas** ( ) const

## Returns

the plot's canvas

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

Redraw the canvas.

## Parameters

<i>painter</i>	Painter used for drawing
<i>canvasRect</i>	Contents rect of the canvas

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

Redraw the canvas items.

## Parameters

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

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

Qt event handler.

Handles QEvent::LayoutRequest and QEvent::PolishRequest

## Parameters

<i>e</i>	Qt Event
----------	----------

## Returns

True, when the event was processed

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

## Returns

`true` if autoscaling is enabled

## Parameters

<i>scaleId</i>	Scale index
----------------	-------------

## See Also

[setAutoScale\(\)](#)

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

Identify the plot item according to an item info object, that has been generated from [itemToInfo\(\)](#).

The default implementation simply tries to unwrap a QwtPlotItem pointer:

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

## Parameters

<i>itemInfo</i>	Plot item
-----------------	-----------

## Returns

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

## See Also

[itemToInfo\(\)](#)

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

Insert a legend.

If the position legend is [QwtPolarPlot::LeftLegend](#) or [QwtPolarPlot::RightLegend](#) the legend will be organized in one column from top to down. Otherwise the legend items will be placed in a table with a best fit number of columns from left to right.

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

Parameters

<i>legend</i>	Legend
<i>pos</i>	The legend's position. For top/left position the number of colums will be limited to 1, otherwise it will be set to unlimited.
<i>ratio</i>	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrinked if it would need more space than the given ratio. The ratio is limited to ]0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See Also

[legend\(\)](#), [QwtPolarLayout::legendPosition\(\)](#), [QwtPolarLayout::setLegendPosition\(\)](#)

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

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

Parameters

<i>plotItem</i>	Plot item
<i>on</i>	Attached/Detached

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

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

The default implementation simply wraps the plot item into a QVariant object. When overloading [itemToInfo\(\)](#) usually [infoToItem\(\)](#) needs to reimplemented too.

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

Parameters

<i>plotItem</i>	Plot item
-----------------	-----------

See Also

[infoToItem\(\)](#)

5.12.4.13 void QwtPolarPlot::layoutChanged ( ) [signal]

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

5.12.4.14 QwtAbstractLegend \* QwtPolarPlot::legend ( )

Returns

the plot's legend

See Also

[insertLegend\(\)](#)

5.12.4.15 `const QwtAbstractLegend * QwtPolarPlot::legend ( ) const`

Returns

the plot's legend

See Also

[insertLegend\(\)](#)

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

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

Parameters

<i>itemInfo</i>	Info about a plot, build from <a href="#">itemToInfo()</a>
-----------------	------------------------------------------------------------

See Also

[itemToInfo\(\)](#), [infoToItem\(\)](#), [QwtAbstractLegend::updateLegend\(\)](#)

5.12.4.17 `const QBrush & QwtPolarPlot::plotBackground ( ) const`

Returns

plot background brush

See Also

[plotBackground\(\)](#), [plotArea\(\)](#)

5.12.4.18 `QwtPolarLayout * QwtPolarPlot::plotLayout ( )`

Returns

Layout, responsible for the geometry of the plot components

5.12.4.19 `const QwtPolarLayout * QwtPolarPlot::plotLayout ( ) const`

Returns

Layout, responsible for the geometry of the plot components

5.12.4.20 `int QwtPolarPlot::plotMarginHint ( ) const`

Returns

Maximum of all item margin hints.

See Also

[QwtPolarItem::marginHint\(\)](#)

5.12.4.21 `QRectF QwtPolarPlot::plotRect ( ) const`

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

## Returns

Bounding rect of the plot area

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

Calculate the bounding rect of the plot area.

The plot area depends on the zoom parameters.

## Parameters

<i>canvasRect</i>	Rectangle of the canvas
-------------------	-------------------------

## Returns

Rectangle for displaying 100% of the plot

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

Redraw the plot.

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

## See Also

[setAutoReplot\(\)](#)

## Warning

Calls [canvas\(\)](#)->`repaint`, take care of infinite recursions

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

Return the scale division of a specified scale.

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

## Parameters

<i>scaleId</i>	Scale index
----------------	-------------

## Returns

Scale division

## See Also

`QwtScaleDiv`, [setScaleDiv\(\)](#), [setScale\(\)](#)

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

Return the scale division of a specified scale.

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

**Parameters**

<i>scaleId</i>	Scale index
----------------	-------------

**Returns**

Scale division

**See Also**

QwtScaleDiv, [setScaleDiv\(\)](#), [setScale\(\)](#)

**5.12.4.26 QwtScaleEngine \* QwtPolarPlot::scaleEngine ( int *scaleId* )****Returns**

Scale engine for a specific scale

**Parameters**

<i>scaleId</i>	Scale index
----------------	-------------

**See Also**

[setScaleEngine\(\)](#)

**5.12.4.27 const QwtScaleEngine \* QwtPolarPlot::scaleEngine ( int *scaleId* ) const****Returns**

Scale engine for a specific scale

**Parameters**

<i>scaleId</i>	Scale index
----------------	-------------

**See Also**

[setScaleEngine\(\)](#)

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

Build a scale map

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

**Parameters**

<i>scaleId</i>	Scale index
<i>radius</i>	Radius of the plot are in pixels

**Returns**

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

**See Also**

QwtScaleMap, [transform\(\)](#), [invTransform\(\)](#)



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

Build a scale map

The azimuth map translates between the scale values and angles from  $[0.0, 2 * \text{PI}]$ . The radial map translates scale values into the distance from the pole. The radial map is calculated from the current geometry of the canvas.

Parameters

<i>scaleId</i>	Scale index
----------------	-------------

Returns

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

See Also

QwtScaleMap, transform(), invTransform()

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

Returns

the maximum number of major ticks for a specified axis

Parameters

<i>scaleId</i>	Scale index
----------------	-------------

See Also

[setScaleMaxMajor\(\)](#)

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

Returns

the maximum number of minor ticks for a specified axis

Parameters

<i>scaleId</i>	Scale index
----------------	-------------

See Also

[setScaleMaxMinor\(\)](#)

5.12.4.32 void QwtPolarPlot::setAutoReplot ( bool *enable* = true )

Set or reset the autoReplot option.

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

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

## Parameters

<i>enable</i>	true or false. Defaults to true.
---------------	----------------------------------

## See Also

[replot\(\)](#)

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

Enable autoscaling.

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

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

## Parameters

<i>scaleId</i>	Scale index
----------------	-------------

## See Also

[hasAutoScale\(\)](#), [setScale\(\)](#), [setScaleDiv\(\)](#), [QwtPolarItem::boundingInterval\(\)](#)

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

Change the origin of the azimuth scale.

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

## Parameters

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

## See Also

[azimuthOrigin\(\)](#)

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

Set the background of the plot area.

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

## Parameters

<i>brush</i>	Background Brush
--------------	------------------

## See Also

[plotBackground\(\)](#), [plotArea\(\)](#)

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

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

## Parameters

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

## See Also

[setScaleMaxMajor\(\)](#), [setAutoScale\(\)](#)

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

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

## Parameters

<i>scaleId</i>	Scale index
<i>scaleDiv</i>	Scale division

## See Also

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

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

Change the scale engine for an axis

## Parameters

<i>scaleId</i>	Scale index
<i>scaleEngine</i>	Scale engine

## See Also

[axisScaleEngine\(\)](#)

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

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

## Parameters

<i>scaleId</i>	Scale index
<i>maxMajor</i>	maximum number of major steps

## See Also

[scaleMaxMajor\(\)](#)

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

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

## Parameters

<i>scaleId</i>	Scale index
<i>maxMinor</i>	maximum number of minor steps

## See Also

[scaleMaxMajor\(\)](#)

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

Change the plot's title

## Parameters

<i>title</i>	New title
--------------	-----------

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

Change the plot's title

## Parameters

<i>title</i>	New title
--------------	-----------

5.12.4.43 `QwtText QwtPolarPlot::title ( ) const`

## Returns

the plot's title

5.12.4.44 `QwtTextLabel * QwtPolarPlot::titleLabel ( )`

## Returns

the plot's title

5.12.4.45 `const QwtTextLabel * QwtPolarPlot::titleLabel ( ) const`

## Returns

the plot's titel label.

5.12.4.46 `void QwtPolarPlot::unzoom ( )`

Unzoom the plot

## See Also

[zoom\(\)](#)

5.12.4.47 `void QwtPolarPlot::updateLegend ( )`

Emit [legendDataChanged\(\)](#) for all plot item

## See Also

[QwtPlotItem::legendData\(\)](#), [legendDataChanged\(\)](#)

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

Emit [legendDataChanged\(\)](#) for a plot item

## Parameters

<i>plotItem</i>	Plot item
-----------------	-----------

## See Also

QwtPlotItem::legendData(), [legendDataChanged\(\)](#)

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

Rebuild the scale

## Parameters

<i>scaleId</i>	Scale index
----------------	-------------

## 5.12.4.50 QwtInterval QwtPolarPlot::visibleInterval ( ) const

## Returns

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

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

Translate and in/decrease the zoom factor.

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

Setting an invalid zoom position disables zooming.

## Parameters

<i>zoomPos</i>	Center of the translation
<i>zoomFactor</i>	Zoom factor

## See Also

[unzoom\(\)](#), [zoomPos\(\)](#), [zoomFactor\(\)](#)

## 5.12.4.52 double QwtPolarPlot::zoomFactor ( ) const

## Returns

Zoom factor

## See Also

[zoom\(\)](#), [zoomPos\(\)](#)

## 5.12.4.53 QwtPointPolar QwtPolarPlot::zoomPos ( ) const

## Returns

Zoom position

## See Also

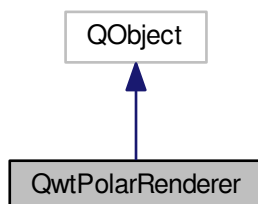
[zoom\(\)](#), [zoomFactor\(\)](#)

### 5.13 QwtPolarRenderer Class Reference

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

```
#include <qwt_polar_renderer.h>
```

Inheritance diagram for QwtPolarRenderer:



#### Public Member Functions

- [QwtPolarRenderer](#) (QObject \*parent=NULL)
- virtual [~QwtPolarRenderer](#) ()
- Destructor.*
- void [renderDocument](#) (QwtPolarPlot \*, const QString &format, const QSizeF &sizeMM, int resolution=85)
- void [renderDocument](#) (QwtPolarPlot \*, const QString &title, const QString &format, const QSizeF &sizeMM, int resolution=85)
- void [renderTo](#) (QwtPolarPlot \*, QPrinter &) const
- Render the plot to a QPrinter.*
- void [renderTo](#) (QwtPolarPlot \*, QPaintDevice &) const
- Render the plot to a QPaintDevice.*
- virtual void [render](#) (QwtPolarPlot \*, QPainter \*, const QRectF &rect) const
- Render the plot to a given rectangle ( f.e QPrinter, QSvgRenderer )*
- bool [exportTo](#) (QwtPolarPlot \*, const QString &documentName, const QSizeF &sizeMM=QSizeF(200, 200), int resolution=85)
- Execute a file dialog and render the plot to the selected file.*
- virtual void [renderTitle](#) (QPainter \*, const QRectF &) const
- virtual void [renderLegend](#) (const QwtPolarPlot \*, QPainter \*, const QRectF &) const

#### 5.13.1 Detailed Description

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

#### 5.13.2 Constructor & Destructor Documentation

##### 5.13.2.1 QwtPolarRenderer::QwtPolarRenderer ( QObject \* parent = NULL ) [explicit]

Constructor

## Parameters

<i>parent</i>	Parent object
---------------	---------------

## 5.13.3 Member Function Documentation

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

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

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

## Parameters

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

## See Also

[renderDocument\(\)](#)

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

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

## Parameters

<i>plot</i>	Plot widget to be rendered
<i>painter</i>	Painter
<i>plotRect</i>	Bounding rectangle for the plot

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

Render a polar plot to a file

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

## Parameters

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

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

Render a plot to a file

Supported formats are:

- pdf
- ps
- svg

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



## Parameters

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

## See Also

[renderTo\(\)](#), [render\(\)](#), [QwtPainter::setRoundingAlignment\(\)](#)

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

Render the legend into a given rectangle.

## Parameters

<i>plot</i>	Plot widget
<i>painter</i>	Painter
<i>rect</i>	Bounding rectangle

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

Render the title into a given rectangle.

## Parameters

<i>painter</i>	Painter
<i>rect</i>	Bounding rectangle

**5.13.3.7** `void QwtPolarRenderer::renderTo ( QwtPolarPlot * plot, QPainter & printer ) const`

Render the plot to a QPainter.

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

## Parameters

<i>plot</i>	Plot to be rendered
<i>printer</i>	Printer to paint on

## See Also

[renderDocument\(\)](#), [render\(\)](#), [QwtPainter::setRoundingAlignment\(\)](#)

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

Render the plot to a `QPaintDevice`.

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

## Parameters

<i>plot</i>	Plot to be rendered
<i>paintDevice</i>	device to paint on, f.e a QImage

#### See Also

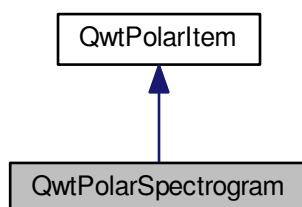
[renderDocument\(\)](#), [render\(\)](#), [QwtPainter::setRoundingAlignment\(\)](#)

## 5.14 QwtPolarSpectrogram Class Reference

An item, which displays a spectrogram.

```
#include <qwt_polar_spectrogram.h>
```

Inheritance diagram for QwtPolarSpectrogram:



#### Public Types

- enum [PaintAttribute](#) { [ApproximatedAtan](#) = 0x01 }
- typedef QFlags< [PaintAttribute](#) > [PaintAttributes](#)  
*Paint attributes.*

#### Public Member Functions

- [QwtPolarSpectrogram](#) ()  
*Constructor.*
- virtual [~QwtPolarSpectrogram](#) ()  
*Destructor.*
- void [setData](#) (QwtRasterData \*[data](#))
- const QwtRasterData \* [data](#) () const
- void [setColorMap](#) (QwtColorMap \*)
- const QwtColorMap \* [colorMap](#) () const
- void [setPaintAttribute](#) ([PaintAttribute](#), bool on=true)
- bool [testPaintAttribute](#) ([PaintAttribute](#)) const
- virtual int [rtti](#) () const
- virtual void [draw](#) (QPainter \*painter, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const
- virtual QwtInterval [boundingInterval](#) (int scaled) const

## Protected Member Functions

- virtual QImage [renderImage](#) (const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, const QRect &rect) const  
*Render an image from the data and color map.*
- virtual void [renderTile](#) (const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, const QPoint &imagePos, const QRect &tile, QImage \*image) const  
*Render a sub-rectangle of an image.*

## 5.14.1 Detailed Description

An item, which displays a spectrogram.

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

## See Also

QwtRasterData, QwtColorMap

## 5.14.2 Member Enumeration Documentation

## 5.14.2.1 enum QwtPolarSpectrogram::PaintAttribute

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

## See Also

[setPaintAttribute\(\)](#), [testPaintAttribute\(\)](#)

## Enumerator

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

## 5.14.3 Member Function Documentation

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

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

## Parameters

<i>scaleId</i>	Scale index
----------------	-------------

## Returns

bounding interval ( == position )

## See Also

[position\(\)](#)

Reimplemented from [QwtPolarItem](#).

#### 5.14.3.2 `const QwtColorMap * QwtPolarSpectrogram::colorMap ( ) const`

##### Returns

Color Map used for mapping the intensity values to colors

##### See Also

[setColorMap\(\)](#)

#### 5.14.3.3 `const QwtRasterData * QwtPolarSpectrogram::data ( ) const`

##### Returns

Spectrogram data

##### See Also

[setData\(\)](#)

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

Draw the spectrogram

##### Parameters

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

Implements [QwtPolarItem](#).

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

Render an image from the data and color map.

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

##### Parameters

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

##### Returns

A QImage::Format\_Indexed8 or QImage::Format\_ARGB32 depending on the color map.

##### See Also

[QwtRasterData::intensity\(\)](#), [QwtColorMap::rgb\(\)](#), [QwtColorMap::colorIndex\(\)](#)

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

Render a sub-rectangle of an image.

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

## Parameters

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

## See Also

[setRenderThreadCount\(\)](#)

## Note

renderTile needs to be reentrant

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

## Returns

[QwtPolarItem::Rtti\\_PolarSpectrogram](#)

Reimplemented from [QwtPolarItem](#).

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

Change the color map

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

## Parameters

<i>colorMap</i>	Color Map
-----------------	-----------

## See Also

[colorMap\(\)](#), [QwtScaleWidget::setColorBarEnabled\(\)](#), [QwtScaleWidget::setColorMap\(\)](#)

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

Set the data to be displayed

## Parameters

<i>data</i>	Spectrogram Data
-------------	------------------

## See Also

[data\(\)](#)

## Warning

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

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

Specify an attribute how to draw the curve

## Parameters

<i>attribute</i>	Paint attribute
<i>on</i>	On/Off

## See Also

[testPaintAttribute\(\)](#)

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

## Parameters

<i>attribute</i>	Paint attribute
------------------	-----------------

## Returns

True, when attribute has been set

## See Also

[setPaintAttribute\(\)](#)

## Index

- ~QwtPolarItemDict
  - QwtPolarItemDict, [43](#)
- activate
  - QwtPolarLayout, [46](#)
- append
  - QwtPolarPicker, [58](#)
- appended
  - QwtPolarPicker, [58](#)
- ApproximatedAtan
  - QwtPolarSpectrogram, [81](#)
- attach
  - QwtPolarItem, [37](#)
- AutoScale
  - QwtPolarItem, [36](#)
- AutoScaling
  - QwtPolarGrid, [26](#)
- autoDelete
  - QwtPolarItemDict, [44](#)
- autoReplot
  - QwtPolarPlot, [65](#)
- axisFont
  - QwtPolarGrid, [27](#)
- axisPen
  - QwtPolarGrid, [27](#)
- azimuthOrigin
  - QwtPolarPlot, [65](#)
- azimuthScaleDraw
  - QwtPolarGrid, [27](#)
- BackingStore
  - QwtPolarCanvas, [12](#)
- backingStore
  - QwtPolarCanvas, [12](#)
- BottomLegend
  - QwtPolarPlot, [64](#)
- boundingInterval
  - QwtPolarCurve, [17](#)
  - QwtPolarItem, [37](#)
  - QwtPolarMarker, [52](#)
  - QwtPolarSpectrogram, [81](#)
- canvas
  - QwtPolarMagnifier, [49](#)
  - QwtPolarPanner, [54](#), [55](#)
  - QwtPolarPicker, [58](#)
  - QwtPolarPlot, [65](#)
- canvasRect
  - QwtPolarLayout, [46](#)
- ClipAxisBackground
  - QwtPolarGrid, [26](#)
- ClipGridLines
  - QwtPolarGrid, [26](#)
- colorMap
  - QwtPolarSpectrogram, [81](#)
- curveFitter
  - QwtPolarCurve, [18](#)
- CurveStyle
  - QwtPolarCurve, [16](#)
- data
  - QwtPolarCurve, [18](#)
  - QwtPolarSpectrogram, [82](#)
- dataSize
  - QwtPolarCurve, [18](#)
- detach
  - QwtPolarItem, [37](#)
- detachItems
  - QwtPolarItemDict, [44](#)
- DisplayFlag
  - QwtPolarGrid, [26](#)
- draw
  - QwtPolarCurve, [18](#)
  - QwtPolarGrid, [27](#)
  - QwtPolarItem, [37](#)
  - QwtPolarMarker, [52](#)
  - QwtPolarSpectrogram, [82](#)
- drawAxis
  - QwtPolarGrid, [28](#)
- drawCanvas
  - QwtPolarPlot, [65](#)
- drawCircles
  - QwtPolarGrid, [28](#)
- drawCurve
  - QwtPolarCurve, [19](#)
- drawItems
  - QwtPolarPlot, [65](#)
- drawLines
  - QwtPolarCurve, [19](#)
- drawRays
  - QwtPolarGrid, [28](#)
- drawSymbols
  - QwtPolarCurve, [19](#)
- end
  - QwtPolarPicker, [58](#)
- event
  - QwtPolarPlot, [66](#)
- exportTo
  - QwtPolarRenderer, [77](#)
- ExternalLegend
  - QwtPolarPlot, [64](#)
- fitCurve
  - QwtPolarFitter, [23](#)
- getUnzoomKey
  - QwtPolarMagnifier, [49](#)
- GridAttribute
  - QwtPolarGrid, [26](#)
- hasAutoScale



- QwtPolarPlot, 66
- HideMaxRadiusLabel
  - QwtPolarGrid, 26
- IgnoreFrames
  - QwtPolarLayout, 46
- IgnoreLegend
  - QwtPolarLayout, 46
- IgnoreScrollbars
  - QwtPolarLayout, 46
- IgnoreTitle
  - QwtPolarLayout, 46
- infoToItem
  - QwtPolarPlot, 66
- insertItem
  - QwtPolarItemDict, 44
- insertLegend
  - QwtPolarPlot, 66
- invTransform
  - QwtPolarCanvas, 12
  - QwtPolarPicker, 60
- invalidate
  - QwtPolarLayout, 46
- isAxisVisible
  - QwtPolarGrid, 28
- isGridVisible
  - QwtPolarGrid, 28
- isMinorGridVisible
  - QwtPolarGrid, 29
- isVisible
  - QwtPolarItem, 38
- itemAttached
  - QwtPolarPlot, 67
- ItemAttribute
  - QwtPolarItem, 36
- itemChanged
  - QwtPolarItem, 38
- itemList
  - QwtPolarItemDict, 44
- itemToInfo
  - QwtPolarPlot, 67
- label
  - QwtPolarMarker, 52
- labelAlignment
  - QwtPolarMarker, 52
- layoutChanged
  - QwtPolarPlot, 67
- layoutLegend
  - QwtPolarLayout, 46
- LeftLegend
  - QwtPolarPlot, 64
- Legend
  - QwtPolarItem, 36
- legend
  - QwtPolarPlot, 67, 68
- LegendShowLine
  - QwtPolarCurve, 17
- LegendShowSymbol
  - QwtPolarCurve, 17
- LegendAttribute
  - QwtPolarCurve, 17
- legendChanged
  - QwtPolarItem, 38
- legendData
  - QwtPolarItem, 38
- legendDataChanged
  - QwtPolarPlot, 68
- legendIcon
  - QwtPolarCurve, 20
  - QwtPolarItem, 38
- legendIconSize
  - QwtPolarItem, 38
- LegendPosition
  - QwtPolarPlot, 64
- legendPosition
  - QwtPolarLayout, 47
- legendRatio
  - QwtPolarLayout, 47
- legendRect
  - QwtPolarLayout, 47
- Lines
  - QwtPolarCurve, 17
- majorGridPen
  - QwtPolarGrid, 29
- marginHint
  - QwtPolarGrid, 29
  - QwtPolarItem, 39
- minorGridPen
  - QwtPolarGrid, 29
- move
  - QwtPolarPicker, 60
- movePlot
  - QwtPolarPanner, 55
- moved
  - QwtPolarPicker, 60
- NoCurve
  - QwtPolarCurve, 17
- Option
  - QwtPolarLayout, 46
- PaintAttribute
  - QwtPolarCanvas, 12
  - QwtPolarSpectrogram, 81
- paintEvent
  - QwtPolarCanvas, 12
- pen
  - QwtPolarCurve, 20
- pickRect
  - QwtPolarPicker, 60
- plot
  - QwtPolarCanvas, 14
  - QwtPolarItem, 39
  - QwtPolarMagnifier, 50
  - QwtPolarPanner, 55

- QwtPolarPicker, 60, 61
- plotBackground
  - QwtPolarPlot, 68
- plotLayout
  - QwtPolarPlot, 68
- plotMarginHint
  - QwtPolarPlot, 68
- plotRect
  - QwtPolarPlot, 68, 69
- position
  - QwtPolarMarker, 52
- QwtPolarCanvas
  - BackingStore, 12
- QwtPolarCurve
  - LegendShowLine, 17
  - LegendShowSymbol, 17
  - Lines, 17
  - NoCurve, 17
  - UserCurve, 17
- QwtPolarGrid
  - AutoScaling, 26
  - ClipAxisBackground, 26
  - ClipGridLines, 26
  - HideMaxRadiusLabel, 26
  - SmartOriginLabel, 26
  - SmartScaleDraw, 26
- QwtPolarItem
  - AutoScale, 36
  - Legend, 36
  - RenderAntialiased, 36
  - Rtti\_PolarCurve, 36
  - Rtti\_PolarGrid, 36
  - Rtti\_PolarItem, 36
  - Rtti\_PolarMarker, 36
  - Rtti\_PolarSpectrogram, 36
  - Rtti\_PolarUserItem, 36
- QwtPolarLayout
  - IgnoreFrames, 46
  - IgnoreLegend, 46
  - IgnoreScrollbars, 46
  - IgnoreTitle, 46
- QwtPolarPlot
  - BottomLegend, 64
  - ExternalLegend, 64
  - LeftLegend, 64
  - RightLegend, 64
  - TopLegend, 64
- QwtPolarSpectrogram
  - ApproximatedAtan, 81
- QwtPolarCanvas, 11
  - backingStore, 12
  - invTransform, 12
  - PaintAttribute, 12
  - paintEvent, 12
  - plot, 14
  - resizeEvent, 14
  - setPaintAttribute, 14
  - testPaintAttribute, 14
  - transform, 14
- QwtPolarCurve, 15
  - boundingInterval, 17
  - curveFitter, 18
  - CurveStyle, 16
  - data, 18
  - dataSize, 18
  - draw, 18
  - drawCurve, 19
  - drawLines, 19
  - drawSymbols, 19
  - LegendAttribute, 17
  - legendIcon, 20
  - pen, 20
  - QwtPolarCurve, 17
  - QwtPolarCurve, 17
  - rtti, 20
  - sample, 20
  - setCurveFitter, 20
  - setData, 21
  - setLegendAttribute, 21
  - setPen, 21
  - setStyle, 21
  - setSymbol, 21
  - style, 22
  - symbol, 22
  - testLegendAttribute, 22
- QwtPolarFitter, 22
  - fitCurve, 23
  - QwtPolarFitter, 23
  - QwtPolarFitter, 23
  - setStepCount, 24
  - stepCount, 24
- QwtPolarGrid, 24
  - axisFont, 27
  - axisPen, 27
  - azimuthScaleDraw, 27
  - DisplayFlag, 26
  - draw, 27
  - drawAxis, 28
  - drawCircles, 28
  - drawRays, 28
  - GridAttribute, 26
  - isAxisVisible, 28
  - isGridVisible, 28
  - isMinorGridVisible, 29
  - majorGridPen, 29
  - marginHint, 29
  - minorGridPen, 29
  - QwtPolarGrid, 27
  - QwtPolarGrid, 27
  - rtti, 30
  - scaleDraw, 30
  - setAxisFont, 30
  - setAxisPen, 30
  - setAzimuthScaleDraw, 31
  - setDisplayFlag, 31
  - setFont, 31

- setGridAttribute, 31
- setMajorGridPen, 31, 32
- setMinorGridPen, 32
- setPen, 32
- setScaleDraw, 32
- showAxis, 33
- showGrid, 33
- showMinorGrid, 33
- testDisplayFlag, 33
- testGridAttribute, 33
- updateScaleDiv, 34
- QwtPolarItem, 34
  - attach, 37
  - boundingInterval, 37
  - detach, 37
  - draw, 37
  - isVisible, 38
  - ItemAttribute, 36
  - itemChanged, 38
  - legendChanged, 38
  - legendData, 38
  - legendIcon, 38
  - legendIconSize, 38
  - marginHint, 39
  - plot, 39
  - QwtPolarItem, 36
  - QwtPolarItem, 36
  - RenderHint, 36
  - renderThreadCount, 39
  - rtti, 39
  - RttiValues, 36
  - setItemAttribute, 39
  - setLegendIconSize, 40
  - setRenderHint, 40
  - setRenderThreadCount, 40
  - setTitle, 40
  - setVisible, 41
  - setZ, 41
  - testItemAttribute, 41
  - testRenderHint, 41
  - title, 42
  - updateScaleDiv, 42
  - z, 42
- QwtPolarItemDict, 42
  - ~QwtPolarItemDict, 43
  - autoDelete, 44
  - detachItems, 44
  - insertItem, 44
  - itemList, 44
  - QwtPolarItemDict, 43
  - QwtPolarItemDict, 43
  - removeItem, 44
  - setAutoDelete, 45
- QwtPolarLayout, 45
  - activate, 46
  - canvasRect, 46
  - invalidate, 46
  - layoutLegend, 46
  - legendPosition, 47
  - legendRatio, 47
  - legendRect, 47
  - Option, 46
  - setLegendPosition, 47
  - setLegendRatio, 48
  - titleRect, 48
- QwtPolarMagnifier, 48
  - canvas, 49
  - getUnzoomKey, 49
  - plot, 50
  - QwtPolarMagnifier, 49
  - QwtPolarMagnifier, 49
  - rescale, 50
  - setUnzoomKey, 50
  - widgetKeyPressEvent, 50
- QwtPolarMarker, 50
  - boundingInterval, 52
  - draw, 52
  - label, 52
  - labelAlignment, 52
  - position, 52
  - rtti, 52
  - setLabel, 53
  - setLabelAlignment, 53
  - setSymbol, 53
  - symbol, 53
- QwtPolarPanner, 53
  - canvas, 54, 55
  - movePlot, 55
  - plot, 55
  - widgetMouseEvent, 55
- QwtPolarPicker, 55
  - append, 58
  - appended, 58
  - canvas, 58
  - end, 58
  - invTransform, 60
  - move, 60
  - moved, 60
  - pickRect, 60
  - plot, 60, 61
  - QwtPolarPicker, 57, 58
  - QwtPolarPicker, 57, 58
  - selected, 61
  - trackerText, 61
  - trackerTextPolar, 61
- QwtPolarPlot, 61
  - autoReplot, 65
  - azimuthOrigin, 65
  - canvas, 65
  - drawCanvas, 65
  - drawItems, 65
  - event, 66
  - hasAutoScale, 66
  - infoToItem, 66
  - insertLegend, 66
  - itemAttached, 67

- itemToInfo, 67
- layoutChanged, 67
- legend, 67, 68
- legendDataChanged, 68
- LegendPosition, 64
- plotBackground, 68
- plotLayout, 68
- plotMarginHint, 68
- plotRect, 68, 69
- QwtPolarPlot, 64, 65
- QwtPolarPlot, 64, 65
- replot, 69
- scaleDiv, 69
- scaleEngine, 70
- scaleMap, 70
- scaleMaxMajor, 71
- scaleMaxMinor, 71
- setAutoReplot, 71
- setAutoScale, 72
- setAzimuthOrigin, 72
- setPlotBackground, 72
- setScale, 72
- setScaleDiv, 73
- setScaleEngine, 73
- setScaleMaxMajor, 73
- setScaleMaxMinor, 73
- setTitle, 74
- title, 74
- titleLabel, 74
- unzoom, 74
- updateLegend, 74
- updateScale, 75
- visibleInterval, 75
- zoom, 75
- zoomFactor, 75
- zoomPos, 75
- QwtPolarRenderer, 76
  - exportTo, 77
  - QwtPolarRenderer, 76
  - QwtPolarRenderer, 76
  - render, 77
  - renderDocument, 77
  - renderLegend, 79
  - renderTitle, 79
  - renderTo, 79
- QwtPolarSpectrogram, 80
  - boundingInterval, 81
  - colorMap, 81
  - data, 82
  - draw, 82
  - PaintAttribute, 81
  - renderImage, 82
  - renderTile, 82
  - rtti, 84
  - setColorMap, 84
  - setData, 84
  - setPaintAttribute, 84
  - testPaintAttribute, 85
- removeItem
  - QwtPolarItemDict, 44
- render
  - QwtPolarRenderer, 77
- RenderAntialiased
  - QwtPolarItem, 36
- renderDocument
  - QwtPolarRenderer, 77
- RenderHint
  - QwtPolarItem, 36
- renderImage
  - QwtPolarSpectrogram, 82
- renderLegend
  - QwtPolarRenderer, 79
- renderThreadCount
  - QwtPolarItem, 39
- renderTile
  - QwtPolarSpectrogram, 82
- renderTitle
  - QwtPolarRenderer, 79
- renderTo
  - QwtPolarRenderer, 79
- replot
  - QwtPolarPlot, 69
- rescale
  - QwtPolarMagnifier, 50
- resizeEvent
  - QwtPolarCanvas, 14
- RightLegend
  - QwtPolarPlot, 64
- rtti
  - QwtPolarCurve, 20
  - QwtPolarGrid, 30
  - QwtPolarItem, 39
  - QwtPolarMarker, 52
  - QwtPolarSpectrogram, 84
- Rtti\_PolarCurve
  - QwtPolarItem, 36
- Rtti\_PolarGrid
  - QwtPolarItem, 36
- Rtti\_PolarItem
  - QwtPolarItem, 36
- Rtti\_PolarMarker
  - QwtPolarItem, 36
- Rtti\_PolarSpectrogram
  - QwtPolarItem, 36
- Rtti\_PolarUserItem
  - QwtPolarItem, 36
- RttiValues
  - QwtPolarItem, 36
- sample
  - QwtPolarCurve, 20
- scaleDiv
  - QwtPolarPlot, 69
- scaleDraw
  - QwtPolarGrid, 30
- scaleEngine
  - QwtPolarPlot, 70

- scaleMap
  - QwtPolarPlot, [70](#)
- scaleMaxMajor
  - QwtPolarPlot, [71](#)
- scaleMaxMinor
  - QwtPolarPlot, [71](#)
- selected
  - QwtPolarPicker, [61](#)
- setAutoDelete
  - QwtPolarItemDict, [45](#)
- setAutoReplot
  - QwtPolarPlot, [71](#)
- setAutoScale
  - QwtPolarPlot, [72](#)
- setAxisFont
  - QwtPolarGrid, [30](#)
- setAxisPen
  - QwtPolarGrid, [30](#)
- setAzimuthOrigin
  - QwtPolarPlot, [72](#)
- setAzimuthScaleDraw
  - QwtPolarGrid, [31](#)
- setColorMap
  - QwtPolarSpectrogram, [84](#)
- setCurveFitter
  - QwtPolarCurve, [20](#)
- setData
  - QwtPolarCurve, [21](#)
  - QwtPolarSpectrogram, [84](#)
- setDisplayFlag
  - QwtPolarGrid, [31](#)
- setFont
  - QwtPolarGrid, [31](#)
- setGridAttribute
  - QwtPolarGrid, [31](#)
- setItemAttribute
  - QwtPolarItem, [39](#)
- setLabel
  - QwtPolarMarker, [53](#)
- setLabelAlignment
  - QwtPolarMarker, [53](#)
- setLegendAttribute
  - QwtPolarCurve, [21](#)
- setLegendIconSize
  - QwtPolarItem, [40](#)
- setLegendPosition
  - QwtPolarLayout, [47](#)
- setLegendRatio
  - QwtPolarLayout, [48](#)
- setMajorGridPen
  - QwtPolarGrid, [31](#), [32](#)
- setMinorGridPen
  - QwtPolarGrid, [32](#)
- setPaintAttribute
  - QwtPolarCanvas, [14](#)
  - QwtPolarSpectrogram, [84](#)
- setPen
  - QwtPolarCurve, [21](#)
  - QwtPolarGrid, [32](#)
- setPlotBackground
  - QwtPolarPlot, [72](#)
- setRenderHint
  - QwtPolarItem, [40](#)
- setRenderThreadCount
  - QwtPolarItem, [40](#)
- setScale
  - QwtPolarPlot, [72](#)
- setScaleDiv
  - QwtPolarPlot, [73](#)
- setScaleDraw
  - QwtPolarGrid, [32](#)
- setScaleEngine
  - QwtPolarPlot, [73](#)
- setScaleMaxMajor
  - QwtPolarPlot, [73](#)
- setScaleMaxMinor
  - QwtPolarPlot, [73](#)
- setStepCount
  - QwtPolarFitter, [24](#)
- setStyle
  - QwtPolarCurve, [21](#)
- setSymbol
  - QwtPolarCurve, [21](#)
  - QwtPolarMarker, [53](#)
- setTitle
  - QwtPolarItem, [40](#)
  - QwtPolarPlot, [74](#)
- setUnzoomKey
  - QwtPolarMagnifier, [50](#)
- setVisible
  - QwtPolarItem, [41](#)
- setZ
  - QwtPolarItem, [41](#)
- showAxis
  - QwtPolarGrid, [33](#)
- showGrid
  - QwtPolarGrid, [33](#)
- showMinorGrid
  - QwtPolarGrid, [33](#)
- SmartOriginLabel
  - QwtPolarGrid, [26](#)
- SmartScaleDraw
  - QwtPolarGrid, [26](#)
- stepCount
  - QwtPolarFitter, [24](#)
- style
  - QwtPolarCurve, [22](#)
- symbol
  - QwtPolarCurve, [22](#)
  - QwtPolarMarker, [53](#)
- testDisplayFlag
  - QwtPolarGrid, [33](#)
- testGridAttribute
  - QwtPolarGrid, [33](#)
- testItemAttribute
  - QwtPolarItem, [41](#)

- testLegendAttribute
  - QwtPolarCurve, [22](#)
- testPaintAttribute
  - QwtPolarCanvas, [14](#)
  - QwtPolarSpectrogram, [85](#)
- testRenderHint
  - QwtPolarItem, [41](#)
- title
  - QwtPolarItem, [42](#)
  - QwtPolarPlot, [74](#)
- titleLabel
  - QwtPolarPlot, [74](#)
- titleRect
  - QwtPolarLayout, [48](#)
- TopLegend
  - QwtPolarPlot, [64](#)
- trackerText
  - QwtPolarPicker, [61](#)
- trackerTextPolar
  - QwtPolarPicker, [61](#)
- transform
  - QwtPolarCanvas, [14](#)
- unzoom
  - QwtPolarPlot, [74](#)
- updateLegend
  - QwtPolarPlot, [74](#)
- updateScale
  - QwtPolarPlot, [75](#)
- updateScaleDiv
  - QwtPolarGrid, [34](#)
  - QwtPolarItem, [42](#)
- UserCurve
  - QwtPolarCurve, [17](#)
- visibleInterval
  - QwtPolarPlot, [75](#)
- widgetKeyPressEvent
  - QwtPolarMagnifier, [50](#)
- widgetMousePressEvent
  - QwtPolarPanner, [55](#)
- z
  - QwtPolarItem, [42](#)
- zoom
  - QwtPolarPlot, [75](#)
- zoomFactor
  - QwtPolarPlot, [75](#)
- zoomPos
  - QwtPolarPlot, [75](#)