Download

Stable Qwt releases are available from the Qwt project page.

Qwt-6.2.0 consists of 4 files:

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

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

- qwtdesigner-6.2.0-*.zip

Installing Qwt

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

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

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

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

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

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

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

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

Configuration

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

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

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

Build and installation

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

Unix-like systems

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

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

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

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

```
make
```

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

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

```
sudo make install
```

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

Windows

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

Creating a makefile usually means something like:

```
cd qwt-6.2.0
qmake qwt.pro
```

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

MinGW

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

```
mingw32-make
```

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

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

```
mingw32-make install
```

MSVC

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

```
nmake
```

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

```
nmake install
```

Qwt and the Qt tool chain

Designer plugin

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

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

The Qt documentation explains all options in detail:

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

F.e. on a Linux system you could add the following lines to .bashrc:

```
QT_PLUGIN_PATH="${QWT_ROOT}/plugins:$QT_PLUGIN_PATH"
```

```
export QT_PLUGIN_PATH
```

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

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

Online Help

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

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

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

For browsing the documentation in the Qt Assistant:

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

Building a Qwt application

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

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

- Adding Qwt as qmake feature

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

  After adding Qwt as a feature f.e on Linux as a persistent property ....

  ```
  qmake -set QMAKEFEATURES ${QWT_ROOT}/features
  ```

  .. the following line can be added to the application project file:

  ```
  CONFIG += qwt
  ```

- Including qwt.prf in the application project file

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

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

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

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

- QMAKE_LFLAGS_RPATH
- QMAKE_RPATH
- QMAKE_RPATHDIR

Running a Qwt application

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

Windows

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

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

GNU/Linux

Read the documentation about:

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

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