Quantum GIS (QGIS)

Building QGIS from source - step by step

Sunday June 12, 2011

Last Updated: Sunday June 12, 2011 Last Change : Sunday June 05, 2011

Contents

1 Introduction

This document is the original installation guide of the described software Quantum GIS. The software and hardware descriptions named in this document are in most cases registered trademarks and are therefore subject to the legal requirements. Quantum GIS is subject to the GNU General Public License. Find more information on the Quantum GIS Homepage: http://www.qgis.org

The details, that are given in this document have been written and verified to the best of knowledge and responsibility of the editors. Nevertheless, mistakes concerning the content are possible. Therefore, all data are not liable to any duties or guarantees. The editors and publishers do not take any responsibility or liability for failures and their consequences. You are always welcome for indicating possible mistakes.

You can download this document as part of the Quantum GIS 'User and Installation Guide' in HTML and PDF format via http://www.qgis.org. A current version is also available at the wiki, see: http://www.qgis.org/wiki/Installation_Guide

Translations of this document can also be downloaded at the documentation area of the Quantum GIS project at http://www.qgis.org. More information is available via http://wiki.qgis.org/qgiswiki/DocumentationWritersCorner.

Please visit http://qgis.org for information on joining our mailing lists and getting involved in the project further.

/!\ **Note to document writers:** Please use this document as the central place for describing build procedures. Please do not remove this notice.

/!\ **Note to document writers:** This documented is generated from doc/INSTALL.t2t - if you need to edit this document, be sure to edit that file rather than the generated INSTALL document found in the root of the source directory.

2 Overview

QGIS, like a number of major projects (eg. KDE 4.0), uses CMake (http://www.cmake.org) for building from source.

Following a summary of the required dependencies for building:

Required build tools:

- CMake >= 2.6.0
- Flex
- Bison

Required build deps:

- Qt >= 4.4.0
- Proj >= 4.4.x
- GEOS >= 3.0
- Sqlite3 >= 3.0.0
- GDAL/OGR >= 1.4.x
- Qwt >= 5.0

Optional dependencies:

- for GRASS plugin - GRASS >= 6.0.0 (libraries compiled with exceptions support on Linux 32bit)
- for georeferencer - GSL >= 1.8
- for postgis support and SPIT plugin - PostgreSQL >= 8.0.x
- for gps plugin - expat >= 1.95 and gpsbabel
- for mapserver export and PyQGIS - Python >= 2.3 (2.5+ preferred)
- for python support - SIP >= 4.8, PyQt >= must match Qt version
- for qgis mapserver - FastCGI

3 Building on GNU/Linux

3.1 Building QGIS with Qt 4.x

Requires: Ubuntu / Debian derived distro

These notes are for Ubuntu - other versions and Debian derived distros may require slight variations in package names.

These notes are for if you want to build QGIS from source. One of the major aims here is to show how this can be done using binary packages for ***all*** dependencies - building only the core QGIS stuff from source. I prefer this approach because it means we can leave the business of managing system packages to apt and only concern ourselves with coding QGIS!

This document assumes you have made a fresh install and have a 'clean' system. These instructions should work fine if this is a system that has already been in use for a while, you may need to just skip those steps which are irrelevant to you.

/!\ **Note:** Refer to the section Building Debian packages for building debian packages. Unless you plan to develop on QGIS, that is probably the easiest option to compile and install QGIS.

3.2 Prepare apt

The packages qgis depends on to build are available in the "universe" component of Ubuntu. This is not activated by default, so you need to activate it:

1. Edit your /etc/apt/sources.list file. 2. Uncomment the all the lines starting with "deb"

Also you will need to be running (K)Ubuntu 'edgy' or higher in order for all dependencies to be met.

Now update your local sources database:

Listing

```
sudo apt-get update
```

3.3 Install build dependencies

Distribution	install command for packages
lenny	apt-get install bison cmake doxygen flex graphviz grass-dev libexpat1-dev libfcgi-dev libgdal1-dev libgeos-dev libgsl0-dev libpq-dev libqt4-
lucid	apt-get install bison cmake doxygen flex graphviz grass-dev libexpat1-dev libfcgi-dev libgdal1-dev libgeos-dev libgsl0-dev libpq-dev libpro
maverick	apt-get install bison cmake doxygen flex graphviz grass-dev libexpat1-dev libfcgi-dev libgdal1-dev libgeos-dev libgsl0-dev libpq-dev libpro
natty	apt-get install bison cmake doxygen flex graphviz grass-dev libexpat1-dev libfcgi-dev libgdal1-dev libgeos-dev libgsl0-dev libpq-dev libpro
sid	apt-get install bison cmake doxygen flex graphviz grass-dev libexpat1-dev libfcgi-dev libgdal1-dev libgeos-dev libgsl0-dev libpq-dev libpro
squeeze	apt-get install bison cmake doxygen flex graphviz grass-dev libexpat1-dev libfcgi-dev libgdal1-dev libgeos-dev libgsl0-dev libpq-dev libpro

(extracted from the respective control files in `debian/`)

/!\ **A Special Note:** If you are following this set of instructions on a system where you already have Qt3 development tools installed, there will be a conflict between Qt3 tools and Qt4 tools. For example, qmake will point to the Qt3 version not the Qt4. Ubuntu Qt4 and Qt3 packages are designed to live alongside each other. This means that for example if you have them both installed you will have three qmake exe's:

Listing

```
/usr/bin/qmake -> /etc/alternatives/qmake
/usr/bin/qmake-qt3
/usr/bin/qmake-qt4
```

The same applies to all other Qt binaries. You will notice above that the canonical 'qmake' is managed by apt alternatives, so before we start to build QGIS, we need to make Qt4 the default. To return Qt3 to default later you can use this same process.

You can use apt alternatives to correct this so that the Qt4 version of applications is used in all cases:

Listing

```
sudo update-alternatives --config qmake
sudo update-alternatives --config uic
sudo update-alternatives --config designer
sudo update-alternatives --config assistant
sudo update-alternatives --config qtconfig
sudo update-alternatives --config moc
sudo update-alternatives --config lupdate
sudo update-alternatives --config lrelease
sudo update-alternatives --config linguist
```

Use the simple command line dialog that appears after running each of the above commands to select the Qt4 version of the relevant applications.

/!\ **Note:** For python language bindings SIP >= 4.5 and PyQt4 >= 4.1 is required! Some stable GNU/Linux distributions (e.g. Debian or SuSE) only provide SIP < 4.5 and PyQt4 < 4.1. To include support for python language bindings you may need to build and install those packages from source.

3.4 Setup ccache (Optional)

You should also setup ccache to speed up compile times:

Listing

```
cd /usr/local/bin
sudo ln -s /usr/bin/ccache gcc
sudo ln -s /usr/bin/ccache g++
```

3.5 Prepare your development environment

As a convention I do all my development work in $HOME/dev/<language>, so in this case we will create a work environment for C++ development work like this:

Listing

```
mkdir -p ${HOME}/dev/cpp
cd ${HOME}/dev/cpp
```

This directory path will be assumed for all instructions that follow.

3.6 Check out the QGIS Source Code

There are two ways the source can be checked out. Use the anonymous method if you do not have edit privaleges for the QGIS source repository, or use the developer checkout if you have permissions to commit source code changes.

1. Anonymous Checkout

Listing

```
cd ${HOME}/dev/cpp
git clone git://github.com/qgis/Quantum-GIS.git
```

2. Developer Checkout

Listing

```
cd ${HOME}/dev/cpp
git clone git@github.com:qgis/Quantum-GIS.git
```


3.7 Starting the compile

I compile my development version of QGIS into my ~/apps directory to avoid conflicts with Ubuntu packages that may be under /usr. This way for example you can use the binary packages of QGIS on your system along side with your development version. I suggest you do something similar:

Listing

```
mkdir -p ${HOME}/apps
```

Now we create a build directory and run ccmake:

Listing

```
cd Quantum-GIS
mkdir build-master
cd build-master
ccmake ..
```

When you run ccmake (note the .. is required!), a menu will appear where you can configure various aspects of the build. If you do not have root access or do not want to overwrite existing QGIS installs (by your packagemanager for example), set the CMAKE_BUILD_PREFIX to somewhere you have write access to (I usually use /home/timlinux/apps). Now press 'c' to configure, 'e' to dismiss any error messages that may appear. and 'g' to generate the make files. Note that sometimes 'c' needs to be pressed several times before the 'g' option becomes available. After the 'g' generation is complete, press 'q' to exit the ccmake interactive dialog.

Now on with the build:

Listing

```
make
make install
```

It may take a little while to build depending on your platform.

After that you can try to run QGIS:

Listing

```
$HOME/apps/bin/qgis
```

If all has worked properly the QGIS application should start up and appear on your screen.

3.8 Building Debian packages

Instead of creating a personal installation as in the previous step you can also create debian package. This is done from the qgis root directory, where you'll find a debian directory.

First you need to install the debian packaging tools once:

Listing

```
apt-get install build-essential
```

First you need to create an changelog entry for your distribution. For example for Ubuntu Lucid:

Listing

```
dch -l ~lucid  --force-distribution --distribution lucid "lucid build"
```

The QGIS packages will be created with:

Listing

```
dpkg-buildpackage -us -uc -b
```

/!\ **Note:** If `dpkg-buildpackage` complains about unmet build dependencies you can install them using `apt-get` and re-run the command.

/!\ **Note:** If you have `libqgis1-dev` installed, you need to remove it first using `dpkg -r libqgis1-dev`. Otherwise `dpkg-buildpackage` will complain about a build conflict.

The packages are created in the parent directory (ie. one level up). Install them using dpkg. E.g.:

Listing

```
sudo debi
```

3.9 A practical case: Building QGIS and GRASS from source on Ubuntu with ECW and MrSID formats support

The following procedure has been tested on Ubuntu 8.04, 8.10 and 9.04 32bit. If you want to use different versions of the software (gdal, grass, qgis), just make the necessary adjustments to the following code. This guide assumes that you don't have installed any previous version of gdal, grass and qgis.

3.9.1 Step 1: install base packages

First you need to install the necessary packages required to download the source code and compile it. Open the terminal and issue the following command:

Listing

```
sudo apt-get install build-essential g++ subversion
```

3.9.2 Step 2: compile and install the ecw libraries

Go to the ERDAS web site http://www.erdas.com/ and follow the links "'products –¿ ECW JPEG2000 Codec SDK –¿ downloads'" then download the "'Image Compression SDK Source Code 3.3'" (you'll need to make a registration and accept a license).

Uncompress the arquive in a proper location (this guide assumes that all the downloaded source code will be placed in the user home) and the enter the newly created folder

Listing

```
cd /libecwj2-3.3
```

Compile the code with the standard commands

Listing

```
./configure
```

then

Listing

```
make
```

then

Listing

```
sudo make install
```

leave the folder

Listing

```
cd ..
```


3.9.3 Step 3: download the MrSID binaries

Go to the LIZARDTECH web site http://www.lizardtech.com/ and follow the links "'download –¿ Developer SDKs"', then download the "'GeoExpress SDK for Linux (x86) - gcc 4.1 32-bit"' (you'll need to make a registration and accept a license).

Uncompress the downloaded file. The resulting directory name should be similar to "Geo_DSDK-7.0.0.2167"

3.9.4 Step 4: compile and install the gdal libraries

Download the latest gdal source code

Listing

```
svn checkout https://svn.osgeo.org/gdal/trunk/gdal gdal
```

then copy a few files from the MrSID binaries folder to the folder with the gdal source code ('replace "USERNAME" with your actual account username')

Listing

```
cp /home/USERNAME/Geo_DSDK-7.0.0.2167/include/*.* /home/USERNAME/gdal/frmts/mrsid/
```

enter the gdal source code folder

Listing

```
cd /gdal
```

and run configure with a few specific parameters

Listing

```
./configure --without-grass --with-mrsid=../Geo_DSDK-7.0.0.2167 --without-jp2mrsid
```

at the end of the configuration process you should read something like

Listing

```
...
GRASS support:            no
...
...
...
ECW support:              yes
MrSID support             yes
...
```


then compile normally

Listing

```
make
```

and

Listing

```
sudo make install
```

finish the process by creating the necessary links to the most recent shared libraries

Listing

```
sudo ldconfig
```

at this point you may want to check if gdal was compiled correctly with MrSID and ECW support by issuing one (or both) of the following commands

Listing

```
gdalinfo --formats | grep 'ECW'
```

Listing

```
gdalinfo --formats | grep 'SID'
```

leave the folder

Listing

```
cd ..
```

3.9.5 Step 5: compile and install GRASS

Before downloading and compile GRASS source code you need to install a few other libraries and programs. We can do this through apt

Listing

```
sudo apt-get install flex bison libreadline5-dev libncurses5-dev lesstif2-dev debhelper dpatch libtiff4-dev \
tcl8.4-dev tk8.4-dev fftw-dev xlibmesa-gl-dev libfreetype6-dev autoconf2.13 autotools-dev \
libgdal1-dev proj libjpeg62-dev libpng12-dev libpq-dev unixodbc-dev doxygen fakeroot cmake \
python-dev python-qt4-common python-qt4-dev python-sip4 python2.5-dev sip4 libglew1.5-dev libxmu6 \
libqt4-dev libgsl0-dev python-qt4 swig python-wxversion python-wxgtk2.8 libwxgtk2.8-0 libwxbase2.8-0 tcl8.4-dev \
tk8.4-dev tk8.4 libfftw3-dev libfftw3-3
```


At this point we can get the GRASS source code: you may want to download it through svn or maybe you want just to download the latest available source code arquive. For example the GRASS 6.4rc4 is available at http://grass.itc.it/grass64/source/grass-6.4.0RC4.tar.gz

Uncompress the arquive, enter the newly created folder and run configure with a few specific parameters

Listing

```
CFLAGS="-fexceptions" ./configure --with-tcltk-includes=/usr/include/tcl8.4 --with-proj-share=/usr/share/proj --with-
--with-python=/usr/bin/python2.5-config
```

The additional gcc option -fexceptions is necessary to enable exceptions support in GRASS libraries. It is currently the only way to avoid QGIS crashes if a fatal error happens in GRASS library. See also http://trac.osgeo.org/grass/ticket/869

Then as usual (it will take a while)

Listing

```
make
```

and

Listing

```
sudo make install
```

leave the folder

Listing

```
cd ..
```

you have now compiled and installed GRASS (also with the new wxpyhton interface) so you may want to give it a try

Listing

```
grass64 -wxpython
```

3.9.6 Step 6: compile and install QGIS

As for GRASS you can obtain the QGIS source code from different sources, for instance from svn or just by downloading one of the source code arquives available at http://www.qgis.org/download/sources.html

For example download the QGIS 1.1.0 source code here http://download.osgeo.org/qgis/src/qgis_1.1.0.tar.gz

uncompress the arquive and enter the newly created folder

Listing

```
cd /qgis_1.1.0
```

then run ccmake

Listing

```
ccmake .
```

press the "c" key, then when the option list will appear we need to manually configure the "GRASS_PREFIX" parameter. Scroll down until the "GRASS_PREFIX" will appear, press enter and manually set it to

Listing

```
/usr/local/grass-6.4.0RC4
```

then press enter again.

Press the "c" again and the option "Press [g] to generate and exit" will appear. Press the "g" key to generate and exit.

then as usual (it will take a while)

Listing

```
make
```

and

Listing

```
sudo make install
```

At the end of the process you should have QGIS and GRASS working with MrSID and ECW raster format support.

To run QGIS just use this command

Listing

```
qgis
```


4 Building on Windows

4.1 Building with Microsoft Visual Studio

This section describes how to build QGIS using Visual Studio on Windows. This is currently also who the binary QGIS packages are made (earlier versions used MinGW).

This section describes the setup required to allow Visual Studio to be used to build QGIS.

4.1.1 Visual C++ Express Edition

The free (as in free beer) Express Edition installer is available under:

http://download.microsoft.com/download/d/c/3/dc3439e7-5533-4f4c-9ba0-8577685b6e7e/vcsetup.exe

The optional products are not necessary. In the process the Windows SDKs for Visual Studio 2008 will also be downloaded and installed.

You also need the Microsoft Windows Server® 2003 R2 Platform SDK (for setupapi):

http://download.microsoft.com/download/f/a/d/fad9efde-8627-4e7a-8812-c351ba099151/PSDK-x86.exe

You only need Microsoft Windows Core SDK / Build Environment (x86 32-Bit).

4.1.2 Other tools and dependencies

Download and install following packages:

Tool	Website
CMake	http://www.cmake.org/files/v2.8/cmake-2.8.4-win32-x86.exe
Flex	http://gnuwin32.sourceforge.net/downlinks/flex.php
Bison	http://gnuwin32.sourceforge.net/downlinks/bison.php
SVN	http://sourceforge.net/projects/win32svn/files/1.6.13/Setup-Subversion-1.6.13.msi/download
or GIT	http://msysgit.googlecode.com/files/Git-1.7.4-preview20110204.exe
OSGeo4W	http://download.osgeo.org/osgeo4w/osgeo4w-setup.exe

OSGeo4W does not only provide ready packages for the current QGIS release and nightly builds of the trunk, but also offers most of the dependencies needs to build it.

For the QGIS build you need to install following packages from OSGeo4W (select *Advanced Installation*):

- expat
- fcgi
- gdal17
- grass
- gsl-devel
- iconv
- pyqt4
- qt4-devel
- qwt5-devel-qt4
- sip

This will also select packages the above packages depend on.

Additionally QGIS also needs the include file `unistd.h`, which normally doesn't exist on Windows. It's shipped with Flex/Bison in `GnuWin32\include` and needs to be copied into the `VC\include` directory of your Visual C++ installation.

Earlier versions of this document also covered how to build all above dependencies. If you're interested in that, check the history of this page in the Wiki or the SVN repository.

4.1.3 Setting up the Visual Studio project with CMake

To start a command prompt with an environment that both has the VC++ and the OSGeo4W variables create the following batch file (assuming the above packages were installed in the default locations):

Listing

```
@echo off
path %SYSTEMROOT%\system32;%SYSTEMROOT%;%SYSTEMROOT%\System32\Wbem;%PROGRAMFILES%\CMake 2.8\bin;%PROGRAMFILES%\subver
set PYTHONPATH=

set VS90COMNTOOLS=%PROGRAMFILES%\Microsoft Visual Studio 9.0\Common7\Tools\
call "%PROGRAMFILES%\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86

set INCLUDE=%INCLUDE%;%PROGRAMFILES%\Microsoft Platform SDK for Windows Server 2003 R2\include
set LIB=%LIB%;%PROGRAMFILES%\Microsoft Platform SDK for Windows Server 2003 R2\lib

set OSGEO4W_ROOT=C:\OSGeo4W
call "%OSGEO4W_ROOT%\bin\o4w_env.bat"

@set GRASS_PREFIX=c:/OSGeo4W/apps/grass/grass-6.4.0
@set INCLUDE=%INCLUDE%;%OSGEO4W_ROOT%\apps\gdal-17\include;%OSGEO4W_ROOT%\include
@set LIB=%LIB%;%OSGEO4W_ROOT%\apps\gdal-17\lib;%OSGEO4W_ROOT%\lib

@cmd
```

Start the batch file and on the command prompt checkout the QGIS source from svn to the source directory `qgis-trunk`:

Listing

```
svn co https://svn.osgeo.org/qgis/trunk/qgis qgis-trunk
```

or using git-svn (from the git shell):

Listing

```
git svn clone --username $USER --revision 15611:HEAD https://svn.osgeo.org/qgis/trunk/qgis
```

Create a 'build' directory somewhere. This will be where all the build output will be generated.

Now run `cmake-gui` and in the *Where is the source code:* box, browse to the top level QGIS directory.

In the *Where to build the binaries:* box, browse to the 'build' directory you created.

Adjust the path to bison and flex so that the shortened `C:/Progra~1` is used rather than `C:/Program Files`.

Verify that the 'BINDINGS_GLOBAL_INSTALL' option is not checked, so that python bindings are placed into the output directory when you run the INSTALL target.

Hit `Configure` to start the configuration and select `Visual Studio 9 2008` and keep `native compilers` and click `Finish`.

The configuration should complete without any further questions and allow you to click `Generate`.

Now close `cmake-gui` and continue on the command prompt by starting `vcexpress`. Use File / Open / Project/Solutions and open the qgis-x.y.z.sln File in your project directory.

Change `Solution Configuration` from `Debug` to `RelWithDebInfo` (Release with Debug Info) or `Release` before you build QGIS using the ALL_BUILD target (otherwise you need debug libraries that are not included).

After the build completed you should install QGIS using the INSTALL target.

Install QGIS by building the INSTALL project. By default this will install to c:\Program Files\qgis<version> (this can be changed by changing the CMAKE_INSTALL_PREFIX variable in cmake-gui).

You will also either need to add all the dependency DLLs to the QGIS install directory or add their respective directories to your PATH.

4.1.4 Packaging

To create a windows 'all in one' standalone package under ubuntu (yes you read correctly) do the following:

Listing

```
sudo apt-get install nsis
```

Now

Listing

```
cd qgis/ms-windows/osgeo4w
```

And run the nsis creation script:

Listing

```
creatensis.pl
```

When the script completes, it should have created a QGIS installer executable in the ms-windows directory (using the QGIS binaries from OSGEO4W).

4.1.5 Packaging your own build of QGIS

Assuming you have completed the above packaging step, if you want to include your own hand built QGIS executables, you need to copy them in from your windows installation into the ms-windows file tree created by the creatensis script.

Listing

```
cd ms-windows/
rm -rf osgeo4w/unpacked/apps/qgis/*
cp -r /tmp/qgis1.7.0/* osgeo4w/unpacked/apps/qgis/
```

Now create a package.

Listing

```
./quickpackage.sh
```

After this you should now have a nsis installer containing your own build of QGIS and all dependencies needed to run it on a windows machine.

4.1.6 Osgeo4w packaging

The actual packaging process is currently not documented, for now please take a look at:

ms-windows/osgeo4w/package.cmd

4.2 Building using MinGW

Note: This section might be outdated as nowadays Visual C++ is use to build the "official" packages.

Note: For a detailed account of building all the dependencies yourself you can visit Marco Pasetti's website here:

http://www.webalice.it/marco.pasetti/qgis+grass/BuildFromSource.html

Read on to use the simplified approach with pre-built libraries...

4.2.1 MSYS

MSYS provides a unix style build environment under windows. We have created a zip archive that contains just about all dependencies.

Get this:

http://download.osgeo.org/qgis/win32/msys.zip

and unpack to c:\msys

If you wish to prepare your msys environment yourself rather than using our pre-made one, detailed instructions are provided elsewhere in this document.

4.2.2 Qt

Download Qt opensource precompiled edition exe and install (including the download and install of mingw) from here:

http://qt.nokia.com/downloads/

When the installer will ask for MinGW, you don't need to download and install it, just point the installer to c:\msys\mingw

When Qt installation is complete:

Edit C:\Qt\4.7.0\bin\qtvars.bat and add the following lines:

Listing

```
set PATH=%PATH%;C:\msys\local\bin;c:\msys\local\lib
set PATH=%PATH%;"C:\Program Files\Subversion\bin"
```

I suggest you also add C:\Qt\4.7.0\bin\ to your Environment Variables Path in the windows system preferences.

If you plan to do some debugging, you'll need to compile debug version of Qt: C:\Qt\4.7.0\bin\qtvars.bat compile_debug

Note: there is a problem when compiling debug version of Qt 4.7, the script ends with this message "mingw32-make: *** No rule to make target 'debug'. Stop.". To compile the debug version you have to go out of src directory and execute the following command:

Listing

```
c:\Qt\4.7.0 make
```

4.2.3 Flex and Bison

Get Flex http://sourceforge.net/project/showfiles.php?group_id=23617&package_id=16424 (the zip bin) and extract it into c:\msys\mingw\bin

4.2.4 Python stuff (optional)

Follow this section in case you would like to use Python bindings for QGIS. To be able to compile bindings, you need to compile SIP and PyQt4 from sources as their installer doesn't include some development files which are necessary.

Download and install Python - use Windows installer

(It doesn't matter to what folder you'll install it)

http://python.org/download/

Download SIP and PyQt4 sources

http://www.riverbankcomputing.com/software/sip/download http://www.riverbankcomputing.com/softwa

Extract each of the above zip files in a temporary directory. Make sure to get versions that match your current Qt installed version.

Compile SIP

Listing

```
c:\Qt\4.7.0\bin\qtvars.bat
python configure.py -p win32-g++
make
make install
```

Compile PyQt

Listing

```
c:\Qt\4.7.0\bin\qtvars.bat
python configure.py
make
make install
```

Final python notes

/!\ You can delete the directories with unpacked SIP and PyQt4 sources after a successfull install, they're not needed anymore.

4.2.5 Subversion

In order to check out QGIS sources from the repository, you need Subversion client. This installer should work fine:

http://www.sliksvn.com/pub/Slik-Subversion-1.6.13-win32.msi

4.2.6 CMake

CMake is build system used by Quantum GIS. Download it from here:

http://www.cmake.org/files/v2.8/cmake-2.8.2-win32-x86.exe

4.2.7 QGIS

Start a cmd.exe window (Start -> Run -> cmd.exe) Create development directory and move into it

Listing

```
md c:\dev\cpp
cd c:\dev\cpp
```

Check out sources from SVN:

For svn trunk:

Listing

```
svn co https://svn.osgeo.org/qgis/trunk/qgis
```

For svn 1.5 branch

Listing

```
svn co https://svn.osgeo.org/qgis/branches/Release-1_5_0 qgis1.5.0
```

4.2.8 Compiling

As a background read the generic building with CMake notes at the end of this document.

Start a cmd.exe window (Start -> Run -> cmd.exe) if you don't have one already. Add paths to compiler and our MSYS environment:

Listing

```
c:\Qt\4.7.0\bin\qtvars.bat
```

For ease of use add c:\Qt\4.7.0\bin\ to your system path in system properties so you can just type qtvars.bat when you open the cmd console. Create build directory and set it as current directory:

Listing

```
cd c:\dev\cpp\qgis
md build
cd build
```


4.2.9 Configuration

Listing

```
cmakesetup ..
```

Note: You must include the '..' above.

Click 'Configure' button. When asked, you should choose 'MinGW Makefiles' as generator.

There's a problem with MinGW Makefiles on Win2K. If you're compiling on this platform, use 'MSYS Makefiles' generator instead.

All dependencies should be picked up automatically, if you have set up the Paths correctly. The only thing you need to change is the installation destination (CMAKE_INSTALL_PREFIX) and/or set 'Debug'.

For compatibility with NSIS packaging scripts I recommend to leave the install prefix to its default c:\program files\

When configuration is done, click 'OK' to exit the setup utility.

4.2.10 Compilation and installation

Listing

```
make make install
```

4.2.11 Run qgis.exe from the directory where it's installed (CMAKE_INSTALL_PREFIX)

Make sure to copy all .dll:s needed to the same directory as the qgis.exe binary is installed to, if not already done so, otherwise QGIS will complain about missing libraries when started.

A possibility is to run qgis.exe when your path contains c:\msys\local\bin and c:\msys\local\lib directories, so the DLLs will be used from that place.

4.2.12 Create the installation package: (optional)

Download and install NSIS from (http://nsis.sourceforge.net/Main_Page)

Now using windows explorer, enter the win_build directory in your QGIS source tree. Read the READMEfile there and follow the instructions. Next right click on qgis.nsi and choose the option 'Compile NSIS Script'.

4.3 Creation of MSYS environment for compilation of Quantum GIS

4.3.1 Initial setup

MSYS

This is the environment that supplies many utilities from UNIX world in Windows and is needed by many dependencies to be able to compile.

Download from here:

http://puzzle.dl.sourceforge.net/sourceforge/mingw/MSYS-1.0.11-2004.04.30-1.exe

Install to `c:\msys`

All stuff we're going to compile is going to get to this directory (resp. its subdirs).

MinGW

Download from here:

http://puzzle.dl.sourceforge.net/sourceforge/mingw/MinGW-5.1.3.exe

Install to `c:\msys\mingw`

It suffices to download and install only `g++` and `mingw-make` components.

Flex and Bison

Flex and Bison are tools for generation of parsers, they're needed for GRASS and also QGIS compilation.

Download the following packages:

http://gnuwin32.sourceforge.net/downlinks/flex-bin-zip.php

http://gnuwin32.sourceforge.net/downlinks/bison-bin-zip.php

http://gnuwin32.sourceforge.net/downlinks/bison-dep-zip.php

Unpack them all to `c:\msys\local`

4.3.2 Installing dependencies

Getting ready

Paul Kelly did a great job and prepared a package of precompiled libraries for GRASS. The package currently includes:

- zlib-1.2.3
- libpng-1.2.16-noconfig
- xdr-4.0-mingw2
- freetype-2.3.4
- fftw-2.1.5
- PDCurses-3.1
- proj-4.5.0
- gdal-1.4.1

It's available for download here:

http://www.stjohnspoint.co.uk/grass/wingrass-extralibs.tar.gz

Moreover he also left the notes how to compile it (for those interested):

http://www.stjohnspoint.co.uk/grass/README.extralibs

Unpack the whole package to `c:\msys\local`

GRASS

Grab sources from CVS or use a weekly snapshot, see:

http://grass.itc.it/devel/cvs.php

In MSYS console go to the directory where you've unpacked or checked out sources (e.g. `c:\msys\local\src\grass-6.3.cvs`)

Run these commands:

Listing

```
export PATH="/usr/local/bin:/usr/local/lib:$PATH"
./configure --prefix=/usr/local --bindir=/usr/local --with-includes=/usr/local/include --with-libs=/usr/local/lib --w
--without-tiff --with-postgres=yes --with-postgres-includes=/local/pgsql/include --with-pgsql-libs=/local/pgsql/lib -
--with-freetype --with-freetype-includes=/mingw/include/freetype2 --without-x --without-tcltk --enable-x11=no --enabl
--with-proj-share=/usr/local/share/proj
make
make install
```

It should get installed to `c:\msys\local\grass-6.3.cvs`

By the way, these pages might be useful:

- http://grass.gdf-hannover.de/wiki/WinGRASS_Current_Status
- http://geni.ath.cx/grass.html

GEOS

Download the sources:

http://geos.refractions.net/geos-2.2.3.tar.bz2

Unpack to e.g. `c:\msys\local\src`

To compile, I had to patch the sources: in file `source/headers/timeval.h` line 13. Change it from:

Listing

```
#ifdef _WIN32
```

to:

Listing

```
#if defined(_WIN32) && defined(_MSC_VER)
```

Now, in MSYS console, go to the source directory and run:

Listing

```
./configure --prefix=/usr/local
make
make install
```


SQLITE

You can use precompiled DLL, no need to compile from source:

Download this archive:

http://www.sqlite.org/sqlitedll-3_3_17.zip

and copy sqlite3.dll from it to `c:\msys\local\lib`

Then download this archive:

http://www.sqlite.org/sqlite-source-3_3_17.zip

and copy sqlite3.h to `c:\msys\local\include`

GSL

Download sources:

ftp://ftp.gnu.org/gnu/gsl/gsl-1.9.tar.gz

Unpack to `c:\msys\local\src`

Run from MSYS console in the source directory:

Listing

```
./configure
make
make install
```

EXPAT

Download sources:

http://dfn.dl.sourceforge.net/sourceforge/expat/expat-2.0.0.tar.gz

Unpack to `c:\msys\local\src`

Run from MSYS console in the source directory:

Listing

```
./configure
```


```
make
make install
```

POSTGRES

We're going to use precompiled binaries. Use the link below for download:

http://wwwmaster.postgresql.org/download/mirrors-ftp?file=%2Fbinary%2Fv8.2.4%2Fwin32%2F
8.2.4-1-binaries-no-installer.zip

copy contents of pgsql directory from the archive to `c:\msys\local`

4.3.3 Cleanup

We're done with preparation of MSYS environment. Now you can delete all stuff in `c:\msys\local\src` - it takes quite a lot of space and it's not necessary at all.

5 Building on MacOS X

In this approach I will try to avoid as much as possible building dependencies from source and rather use frameworks wherever possible.

The base system here is Mac OS X 10.4 (Tiger), with a single architecture build. Included are notes for building on Mac OS X 10.5 (Leopard) and 10.6 (Snow Leopard). Make sure to read each section completely before typing the first command you see.

General note on Terminal usage: When I say "cd" to a folder in a Terminal, it means type "cd " (without the quotes, make sure to type a space after) and then type the path to said folder, then <return>. A simple way to do this without having to know and type the full path is, after type the "cd " part, drag the folder (use the icon in its window title bar, or drag a folder from within a window) from the Desktop to the Terminal, then tap <return>.

Parallel Compilation: On multiprocessor/multicore Macs, it's possible to speed up compilation, but it's not automatic. Whenever you type "make" (but NOT "make install"), instead type:

Listing

```
make -j [n]
```

Replace [n] with the number of cores and/or processors your Mac has. On recent models with hyperthreading processors this can be double the physical count of processors and cores.

ie: Mac Pro "8 Core" model (2 quad core processors) = 8

ie: Macbook Pro i5 (hyperthreading) = 2 cores X 2 = 4

5.1 Install Qt4 from disk image

You need a minimum of Qt-4.4.0. I suggest getting the latest. There is no need for the full Qt SDK, so save yourself some download time and get the frameworks only.

Snow Leopard note: If you are building on Snow Leopard, you will need to decide between 32-bit support in the older, Qt Carbon branch, or 64-bit support in the Qt Cocoa branch. Appropriate installers are available for both as of Qt-4.5.2. Qt 4.6+ is recommended for Cocoa.

PPC note: The readymade Qt Cocoa installers don't include PPC support, you'd have to compile Qt yourself. But, there appear to be issues with Qt Cocoa on PPC Macs anyways. Qt Carbon is recommended on PPC Macs.

http://qt.nokia.com/downloads

If you want debug frameworks, Qt also provides a separate download with these. These are in addition to the non-debug frameworks.

Once downloaded open the disk image and run the installer. Note you need admin privileges to install.

Qt note: Starting in Qt 4.4, libQtCLucene was added, and in 4.5 libQtUiTools was added, both in /usr/lib. When using a system SDK these libraries will not be found. To fix this problem, add symlinks to /usr/local:

Listing

```
sudo ln -s /usr/lib/libQtUiTools.a /usr/local/lib/
sudo ln -s /usr/lib/libQtCLucene.dylib /usr/local/lib/
```

These should then be found automatically on Leopard and above. Earlier systems may need some help by adding '-L/usr/local/lib' to CMAKE_SHARED_LINKER_FLAGS, CMAKE_MODULE_LINKER_FLAGS and CMAKE_EXE_LINKER_FLAGS in the cmake build.

5.2 Install development frameworks for QGIS dependencies

Download William Kyngesburye's excellent GDAL Complete package that includes PROJ, GEOS, GDAL, SQLite3, Spatialite, and image libraries, as frameworks. There is also a GSL framework.

http://www.kyngchaos.com/wiki/software/frameworks

Once downloaded, open and install the frameworks.

William provides an additional installer package for Postgresql (for PostGIS support). Qgis just needs the libpq client library, so unless you want to setup the full Postgres + PostGIS server, all you need is the client-only package. It's available here:

http://www.kyngchaos.com/wiki/software/postgres

Also available is a GRASS application:

http://www.kyngchaos.com/wiki/software/grass

5.2.1 Additional Dependencies: General compatibility note

There are some additional dependencies that, at the time of writing, are not provided as frameworks or installers so we will need to build these from source. If you are wanting to build Qgis as a 64-bit application, you will need to provide the appropriate build commands to produce 64-bit support in dependencies. Likewise, for 32-bit support on Snow Leopard, you will need to override the default system architecture, which is 64-bit, according to instructions for individual dependency packages.

Stable release versions are preferred. Beta and other development versions may have problems and you are on your own with those.

5.2.2 Additional Dependencies: Expat

Snow Leopard note: Snow Leopard includes a usable expat, so this step is not necessary on Snow Leopard.

Get the expat sources:

http://sourceforge.net/project/showfiles.php?group_id=10127

Double-click the source tarball to unpack, then, in Terminal.app, cd to the source folder and:

Listing

```
./configure
make
sudo make install
```

5.2.3 Additional Dependencies: Python

Leopard and Snow Leopard note: Leopard and Snow Leopard include a usable Python 2.5 and 2.6, respectively. So there is no need to install Python on Leopard and Snow Leopard. You can still install Python from python.org if preferred.

If installing from python.org, make sure you install at least the latest Python 2.x from

http://www.python.org/download/

Python 3 is a major change, and may have compatibility issues, so try it at your own risk.

5.2.4 Additional Dependencies: SIP

Retrieve the python bindings toolkit SIP from

http://www.riverbankcomputing.com/software/sip/download

Double-click the source tarball to unpack it, then, in Terminal.app, cd to the source folder. Then for your chosen Python:

python.org Python

Listing

```
python configure.py
make
sudo make install
```

Leopard system Python

SIP wants to install in the system path – this is not a good idea. More configuration is needed to install outside the system path:

Listing

```
python configure.py -n -d /Library/Python/2.5/site-packages -b /usr/local/bin \
-e /usr/local/include -v /usr/local/share/sip -s MacOSX10.5.sdk
```

Snow Leopard system Python

Similar to Leopard, you should install outside the system Python path. Also, you need to specify the architecture you want (requires at least SIP 4.9), and make sure to run the versioned python binary (this one responds to the 'arch' command, 'python' does not).

If you are using 32-bit Qt (Qt Carbon):

Listing

```
python2.6 configure.py -n -d /Library/Python/2.6/site-packages -b /usr/local/bin \
-e /usr/local/include -v /usr/local/share/sip --arch=i386 -s MacOSX10.6.sdk
```

For 64-bit Qt (Qt Cocoa), use this configure line:

Listing

```
python2.6 configure.py -n -d /Library/Python/2.6/site-packages -b /usr/local/bin \
-e /usr/local/include -v /usr/local/share/sip --arch=x86_64 -s MacOSX10.6.sdk
```

continue...

Then continue with compilation and installation:

Listing

```
make
sudo make install
```

5.2.5 Additional Dependencies: PyQt

Retrieve the python bindings toolkit for Qt from

http://www.riverbankcomputing.com/software/pyqt/download

Double-click the source tarball to unpack it, then, in Terminal.app, cd to the source folder. Then for your chosen Python:

python.org Python

Listing

```
python configure.py
yes
```

Leopard system Python

PyQt wants to install in the system path – this is not a good idea. More configuration is needed to install outside the system path:

Listing

```
python configure.py -d /Library/Python/2.5/site-packages -b /usr/local/bin
```

Snow Leopard system Python

Similar to Leopard, you should install outside the system Python path. Also, you need to specify the architecture you want (requires at least PyQt 4.6), and make sure to run the versioned python binary (this one responds to the 'arch' command, which is important for pyuic4, 'python' does not).

If you are using 32-bit Qt (Qt Carbon):

Listing

```
python2.6 configure.py -d /Library/Python/2.6/site-packages -b /usr/local/bin --use-arch i386
```

For 64-bit Qt (Qt Cocoa), use this configure line:

Listing

```
python2.6 configure.py -d /Library/Python/2.6/site-packages -b /usr/local/bin --use-arch x86_64
```

continue...

There is a problem with the configuration that needs to be fixed now (it affects PyQwt compilation later). Edit pyqtconfig.py and change the qt_dir line to:

Listing

```
    'qt_dir': '/usr',
```

Then continue with compilation and installation (this is a good place to use parallel compilation, if you can):

Listing

```
make
sudo make install
```

If there is a problem with undefined symbols in QtOpenGL on Leopard, edit QtOpenGL/makefile and add -undefined dynamic_lookup to LFLAGS. Then make again.

5.2.6 Additional Dependencies: Qwt/PyQwt

The GPS tracking feature uses Qwt. Some popular 3rd-party plugins use PyQwt. You can take care of both with the PyQwt source from:

http://pyqwt.sourceforge.net/

Double-click the tarball to unpack it. The following assumes PyQwt v5.2.0 (comes with Qwt 5.2.1). Normal compilation does both Qwt and PyQwt at the same time, but Qwt is statically linked into PyQwt, and Qgis can't use it. So, we need to split the build.

Now, cd into the qwt-5.2 subdir in a Terminal. Type these commands to build and install:

Listing

```
cat >> qwtconfig.pri <<EOF
CONFIG += release QwtDll
EOF
qmake -spec macx-g++
make
sudo make install
sudo install_name_tool -id /usr/local/qwt-5.2.1-svn/lib/libqwt.5.dylib \
/usr/local/qwt-5.2.1-svn/lib/libqwt.5.dylib
```


The Qwt shared library is now installed in /usr/local/qwt-5.x.x[-svn] (x.x is the minor.point version, and it may be an SVN version). Remember this for QGIS and PyQwt configuration.

Now for PyQwt. Still in the Terminal (for all Pythons, except see Snow Leopard Carbon note below):

Listing

```
cd ../configure
python configure.py --extra-include-dirs=/usr/local/qwt-5.2.1-svn/include \
--extra-lib-dirs=/usr/local/qwt-5.2.1-svn/lib --extra-libs=qwt
make
sudo make install
```

Make sure to use the qwt install path from the Qwt build above.

Snow Leopard note

If using Qt Carbon, you need to specify which architectures to build, otherwise it will default to a combination that does not work (ie x86_64 for a Carbon Qt). This is not needed for Qt Cocoa. Configure as follows:

Listing

```
python configure.py --extra-cflags="-arch i386" --extra-cxxflags="-arch i386" \
--extra-lflags="-arch i386" --extra-include-dirs=/usr/local/qwt-5.2.1-svn/include \
--extra-lib-dirs=/usr/local/qwt-5.2.1-svn/lib --extra-libs=qwt
```

5.2.7 Additional Dependencies: Bison

Leopard and Snow Leopard note: Leopard and Snow Leopard include Bison 2.3, so this step can be skipped on Leopard and Snow Leopard.

The version of bison available by default on Mac OS X 10.4 is too old so you need to get a more recent one on your system. Download at least version 2.3 from:

Listing

```
ftp.gnu.org/gnu/bison/
```

Now build and install it to a prefix of /usr/local.Ê Double-click the source tarball to unpack it, then cd to the source folder and:

Listing

```
./configure --prefix=/usr/local
make
```


```
sudo make install
```

5.3 Install CMake for OSX

Get the latest source release from here:

http://www.cmake.org/cmake/resources/software.html

Binary installers are available for OS X, but they are not recommended (2.4 versions install in /usr instead of /usr/local, and 2.6+ versions are a strange application). Instead, download the source, double-click the source tarball, then cd to the source folder and:

Listing

```
./bootstrap --docdir=/share/doc/CMake --mandir=/share/man
make
sudo make install
```

5.4 QGIS source

Unzip the QGIS source tarball to a working folder of your choice (/usr/somewhere is not a good choice as it's hidden and requires root privileges). If you are reading this from the source, you've already done this.

If you want to experiment with the latest development sources, see the CODING document.

5.5 Configure the build

CMake supports out of source build so we will create a 'build' dir for the build process. OS X uses ${HOME}/Applications as a standard user app folder (it gives it the system app folder icon). If you have the correct permissions you may want to build straight into your /Applications folder. The instructions below assume you are building into a pre-existing ${HOME}/Applications directory. In a Terminal cd to the qgis source folder previously downloaded, then:

Listing

```
mkdir build
cd build
cmake -D CMAKE_INSTALL_PREFIX=~/Applications \
-D CMAKE_BUILD_TYPE=MinSizeRel \
-D WITH_INTERNAL_SPATIALITE=FALSE -D WITH_MAPSERVER=TRUE \
-D QWT_LIBRARY=/usr/local/qwt-5.2.1-svn/lib/libqwt.dylib \
```


```
-D QWT_INCLUDE_DIR=/usr/local/qwt-5.2.1-svn/include \
..
```

This will automatically find and use the previously installed frameworks, and the GRASS application if installed.

Or, to use a Unix-style build of GRASS, use the following cmake invocation (minimum GRASS version as stated in the Qgis requirements, substitute the GRASS path and version as required):

Listing

```
cmake -D CMAKE_INSTALL_PREFIX=~/Applications -D CMAKE_BUILD_TYPE=Release \
-D CMAKE_BUILD_TYPE=MinSizeRel \
-D WITH_INTERNAL_SPATIALITE=FALSE -D WITH_MAPSERVER=TRUE \
-D QWT_LIBRARY=/usr/local/qwt-5.2.1-svn/lib/libqwt.dylib \
-D QWT_INCLUDE_DIR=/usr/local/qwt-5.2.1-svn/include \
-D GRASS_PREFIX=/user/local/grass-6.4.1 \
..
```

Snow Leopard note: To handle 32-bit Qt (Carbon), create a 32bit python wrapper script and add arch flags to the configuration:

Listing

```
sudo cat >/usr/local/bin/python32 <<EOF
#!/bin/sh
exec arch -i386 /usr/bin/python2.6 \${1+"\$@"}
EOF

sudo chmod +x /usr/local/bin/python32

cmake -D CMAKE_INSTALL_PREFIX=~/Applications -D \
-D CMAKE_BUILD_TYPE=MinSizeRel \
-D WITH_INTERNAL_SPATIALITE=FALSE -D WITH_MAPSERVER=TRUE \
-D QWT_LIBRARY=/usr/local/qwt-5.2.1-svn/lib/libqwt.dylib \
-D QWT_INCLUDE_DIR=/usr/local/qwt-5.2.1-svn/include \
-D CMAKE_OSX_ARCHITECTURES=i386 -D PYTHON_EXECUTABLE=/usr/local/bin/python32 \
..
```

Bundling note: Older Qt versions may have problems with some Qt plugins and Qgis. The way to handle this is to bundle Qt inside the Qgis application. You can do this now or wait to see if there are immediate crashes when running Qgis. It's also a good idea to bundle Qt if you need to copy Qgis to other Macs (where you would have to install Xcode just so Qt would install!).

To bundle Qt, add the following line before the last line (the ".." line) in the above cmake configurations:

Listing

```
-D QGIS_MACAPP_BUNDLE=1 \
```

Even better for distribution purposes, to also bundle any extra non-framework, non-standard, libs (ie postgres' libpq) bump the bundle number to 2:

Listing

```
-D QGIS_MACAPP_BUNDLE=2 \
```

5.6 Building

Now we can start the build process (remember the parallel compilation note at the beginning, this is a good place to use it, if you can):

Listing

```
make
```

If all built without errors you can then install it:

Listing

```
make install
```

or, for an /Applications build:

Listing

```
sudo make install
```


6 Authors and Acknowledgments

The following people have contributed to this document:

- Windows MINGW Section
  - Tim Sutton, Godofredo Contreras 2006
  - CMake additions Magnus Homann 2007
  - Python additions Martin Dobias 2007
  - With thanks to Tisham Dhar for preparing the initial msys environment
- Windows MSVC Section (Detailed install)
  - David Willis 2007
  - MSVC install additions Tim Sutton 2007
  - PostgreSQL, Qt compile, SIP, Python, AutoExp additions Juergen Fischer 2007
- Windows MSVC Section (Simplified install)
  - Tim Sutton 2007
  - Juergen Fischer 2007
  - Florian Hillen 2010
- OSX Section
  - Tim Sutton 2007
  - With special thanks to Tom Elwertowski and William Kyngesburye
- GNU/Linux Section
  - Tim Sutton 2006
  - Debian package section: Juergen Fischer 2008