



Quantum GIS

User, Installation and Coding
Guide

Version 1.0.0 'Kore'

Preamble

This document is the original user, installation and coding guide of the described software Quantum GIS. The software and hardware described 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://qgis.osgeo.org>.

The details, data, results etc. in this document have been written and verified to the best of knowledge and responsibility of the authors and editors. Nevertheless, mistakes concerning the content are possible.

Therefore, all data are not liable to any duties or guarantees. The authors, editors and publishers do not take any responsibility or liability for failures and their consequences. You are always welcome to indicate possible mistakes.

This document has been typeset with \LaTeX . It is available as \LaTeX source code via [subversion](#) and online as PDF document via <http://qgis.osgeo.org/documentation/manuals.html>. Translated versions of this document can be downloaded via the documentation area of the QGIS project as well. For more information about contributing to this document and about translating it, please visit: <http://wiki.qgis.org/qgiswiki/DocumentationWritersCorner>

Links in this Document

This document contains internal and external links. Clicking on an internal link moves within the document, while clicking on an external link opens an internet address. In PDF form, internal links are shown in blue, while external links are shown in red and are handled by the system browser. In HTML form, the browser displays and handles both identically.

User, Installation and Coding Guide Authors and Editors:

Tara Athan	Radim Blazek	Godofredo Contreras
Otto Dassau	Martin Dobias	Jürgen E. Fischer
Stephan Holl	Marco Hugentobler	Magnus Homann
Lars Luthman	Gavin Macaulay	Werner Macho
Tyler Mitchell	Brendan Morely	Gary E. Sherman
Tim Sutton	David Willis	

With thanks to Tisham Dhar for preparing the initial msys (MS Windows) environment documentation, to Tom Elwertowski and William Kyngesburye for help in the MAC OSX Installation Section and to Carlos Dávila, Paolo Cavallini and Christian Gunning for revisions. If we have neglected to mention any contributors, please accept our apologies for this oversight.

Copyright © 2004 - 2009 Quantum GIS Development Team

Internet: <http://qgis.osgeo.org>

Contents

Title	i
Preamble	ii
Table of Contents	iii
List of Figures	xii
List of Tables	xiv
List of QGIS Tips	xv
1 Forward	1
1.1 Features	1
1.2 Conventions	4
2 Introduction To GIS	6
2.1 Why is all this so new?	6
2.1.1 Raster Data	7
2.1.2 Vector Data	7
3 Getting Started	9
3.1 Installation	9
3.2 Sample Data	9
3.3 Sample Session	10
4 Features at a Glance	12
4.1 Starting and Stopping QGIS	12
4.1.1 Command Line Options	12
4.2 QGIS GUI	13
4.2.1 Menu Bar	14
4.2.2 Toolbars	17
4.2.3 Map Legend	17
4.2.4 Map View	19
4.2.5 Map Overview	19
4.2.6 Status Bar	20
4.3 Rendering	20
4.3.1 Scale Dependent Rendering	20
4.3.2 Controlling Map Rendering	21
4.4 Measuring	22
4.4.1 Measure length and areas	22
4.5 Projects	22

4.6	Output	23
4.7	GUI Options	24
4.8	Spatial Bookmarks	25
4.8.1	Creating a Bookmark	26
4.8.2	Working with Bookmarks	26
4.8.3	Zooming to a Bookmark	26
4.8.4	Deleting a Bookmark	26
5	Working with Vector Data	27
5.1	ESRI Shapefiles	27
5.1.1	Loading a Shapefile	27
5.1.2	Improving Performance	28
5.1.3	Loading a MapInfo Layer	29
5.1.4	Loading an ArcInfo Coverage	30
5.2	PostGIS Layers	30
5.2.1	Creating a stored Connection	30
5.2.2	Loading a PostGIS Layer	31
5.2.3	Some details about PostgreSQL layers	32
5.2.4	Importing Data into PostgreSQL	32
5.2.5	Improving Performance	34
5.3	The Vector Properties Dialog	35
5.3.1	General Tab	36
5.3.2	Symbology Tab	36
5.3.3	Metadata Tab	38
5.3.4	Labels Tab	38
5.3.5	Actions Tab	40
5.3.6	Attributes Tab	43
5.4	Editing	44
5.4.1	Setting the Snapping Tolerance and Search Radius	45
5.4.2	Topological editing	46
5.4.3	Editing an Existing Layer	46
5.4.4	Creating a New Layer	53
5.5	Query Builder	53
5.6	Select by query	55
6	Working with Raster Data	56
6.1	What is raster data?	56
6.2	Loading raster data in QGIS	56
6.3	Raster Properties Dialog	57
6.3.1	Symbology Tab	58
6.3.2	Transparency Tab	59
6.3.3	Colormap	60

6.3.4	General Tab	61
6.3.5	Metadata Tab	61
6.3.6	Pyramids Tab	61
6.3.7	Histogram Tab	62
7	Working with OGC Data	63
7.1	What is OGC Data	63
7.2	WMS Client	63
7.2.1	Overview of WMS Support	63
7.2.2	Selecting WMS Servers	64
7.2.3	Loading WMS Layers	65
7.2.4	Using the Identify Tool	67
7.2.5	Viewing Properties	67
7.2.6	WMS Client Limitations	68
7.3	WFS Client	69
7.3.1	Loading a WFS Layer	69
8	Working with Projections	71
8.1	Overview of Projection Support	71
8.2	Specifying a Projection	71
8.3	Define On The Fly (OTF) Projection	72
8.4	Custom Coordinate Reference System	74
9	GRASS GIS Integration	76
9.1	Starting the GRASS plugin	76
9.2	Loading GRASS raster and vector layers	77
9.3	GRASS LOCATION and MAPSET	78
9.3.1	Creating a new GRASS LOCATION	78
9.3.2	Adding a new MAPSET	80
9.4	Importing data into a GRASS LOCATION	81
9.5	The GRASS vector data model	82
9.6	Creating a new GRASS vector layer	82
9.7	Digitizing and editing a GRASS vector layer	83
9.8	The GRASS region tool	87
9.9	The GRASS toolbox	87
9.9.1	Working with GRASS modules	87
9.9.2	Working with the GRASS LOCATION browser	89
9.9.3	Customizing the GRASS Toolbox	90
10	Print Composer	92
10.1	Using Print Composer	92
10.1.1	Adding a current QGIS map canvas to the Print Composer	94
10.1.2	Adding other elements to the Print Composer	95

10.1.3	Navigation tools	96
10.1.4	Creating Output	97
11	QGIS Plugins	98
11.1	Managing Plugins	98
11.1.1	Loading a QGIS Core Plugin	98
11.1.2	Loading an external QGIS Plugin	98
11.1.3	Using the QGIS Python Plugin Installer	99
11.2	Data Providers	101
12	Using QGIS Core Plugins	102
12.1	Coordinate Capture Plugin	103
12.2	Decorations Plugins	104
12.2.1	Copyright Label Plugin	104
12.2.2	North Arrow Plugin	105
12.2.3	Scale Bar Plugin	105
12.3	Delimited Text Plugin	107
12.4	Dxf2Shp Converter Plugin	109
12.5	Georeferencer Plugin	110
12.6	Quick Print Plugin	114
12.7	GPS Plugin	115
12.7.1	What is GPS?	115
12.7.2	Loading GPS data from a file	115
12.7.3	GPSBabel	115
12.7.4	Importing GPS data	116
12.7.5	Downloading GPS data from a device	116
12.7.6	Uploading GPS data to a device	117
12.7.7	Defining new device types	118
12.8	Graticule Creator Plugin	120
12.9	Interpolation Plugin	121
12.10	MapServer Export Plugin	123
12.10.1	Creating the Project File	123
12.10.2	Creating the Map File	124
12.10.3	Testing the Map File	126
12.11	OGR Converter Plugin	127
13	Using external QGIS Python Plugins	128
14	Writing a QGIS Plugin in C++	129
14.1	Why C++ and what about licensing	129
14.2	Programming a QGIS C++ Plugin in four steps	129
14.3	Further information	147

15 Writing a QGIS Plugin in Python	148
15.1 Why Python and what about licensing	148
15.2 What needs to be installed to get started	148
15.3 Programming a simple PyQGIS Plugin in four steps	149
15.4 Committing the plugin to repository	152
15.5 Further information	152
16 Creating C++ Applications	154
16.1 Creating a simple mapping widget	154
16.2 Working with QgsMapCanvas	158
17 Creating PyQGIS Applications	161
17.1 Designing the GUI	161
17.2 Creating the MainWindow	162
17.3 Finishing Up	166
17.4 Running the Application	167
18 Help and Support	169
18.1 Mailinglists	169
18.2 IRC	170
18.3 BugTracker	170
18.4 Blog	171
18.5 Wiki	171
A Supported Data Formats	172
A.1 Supported OGR Formats	172
A.2 GDAL Raster Formats	172
B GRASS Toolbox modules	175
B.1 GRASS Toolbox data import and export modules	175
B.2 GRASS Toolbox data type conversion modules	176
B.3 GRASS Toolbox region and projection configuration modules	177
B.4 GRASS Toolbox raster data modules	178
B.5 GRASS Toolbox vector data modules	182
B.6 GRASS Toolbox imagery data modules	185
B.7 GRASS Toolbox database modules	186
B.8 GRASS Toolbox 3D modules	187
B.9 GRASS Toolbox help modules	187
C Installation Guide	188
C.1 General Build Notes	188
C.2 An overview of the dependencies required for building	188
D Building under windows using msys	189

D.1	MSYS:	189
D.2	Qt4.3	189
D.3	Flex and Bison	190
D.4	Python stuff: (optional)	190
D.4.1	Download and install Python - use Windows installer	190
D.4.2	Download SIP and PyQt4 sources	190
D.4.3	Compile SIP	191
D.4.4	Compile PyQt	191
D.4.5	Final python notes	191
D.5	Subversion:	191
D.6	CMake:	191
D.7	QGIS:	191
D.8	Compiling:	192
D.9	Configuration	192
D.10	Compilation and installation	193
D.11	Run qgis.exe from the directory where it's installed (CMAKE_INSTALL_PREFIX)	193
D.12	Create the installation package: (optional)	193
E	Building on Mac OSX using frameworks and cmake (QGIS > 0.8)	193
E.1	Install XCODE	194
E.2	Install Qt4 from .dmg	194
E.3	Install development frameworks for QGIS dependencies	195
E.3.1	Additional Dependencies : GSL	195
E.3.2	Additional Dependencies : Expat	195
E.3.3	Additional Dependencies : SIP	196
E.3.4	Additional Dependencies : PyQt	196
E.3.5	Additional Dependencies : Bison	197
E.4	Install CMAKE for OSX	198
E.5	Install subversion for OSX	198
E.6	Check out QGIS from SVN	199
E.7	Configure the build	200
E.8	Building	201
F	Building on GNU/Linux	201
F.1	Building QGIS with Qt4.x	201
F.2	Prepare apt	201
F.3	Install Qt4	202
F.4	Install additional software dependencies required by QGIS	202
F.5	GRASS Specific Steps	203
F.6	Setup ccache (Optional)	203
F.7	Prepare your development environment	203
F.8	Check out the QGIS Source Code	204


















F.9	Starting the compile	204
F.10	Building Debian packages	205
F.11	Running QGIS	206
G	Creation of MSYS environment for compilation of Quantum GIS	206
G.1	Initial setup	206
G.1.1	MSYS	206
G.1.2	MinGW	206
G.1.3	Flex and Bison	207
G.2	Installing dependencies	207
G.2.1	Getting ready	207
G.2.2	GDAL level one	208
G.2.3	GRASS	209
G.2.4	GDAL level two	210
G.2.5	GEOS	211
G.2.6	SQLITE	211
G.2.7	GSL	212
G.2.8	EXPAT	212
G.2.9	POSTGRES	212
G.3	Cleanup	213
H	Building with MS Visual Studio	213
H.1	Setup Visual Studio	213
H.1.1	Express Edition	213
H.1.2	All Editions	214
H.2	Download/Install Dependencies	214
H.2.1	Flex and Bison	214
H.2.2	To include PostgreSQL support in Qt	214
H.2.3	Qt	215
H.2.4	Proj.4	215
H.2.5	GSL	216
H.2.6	GEOS	216
H.2.7	GDAL	217
H.2.8	PostGIS	217
H.2.9	Expat	217
H.2.10	CMake	218
H.3	Building QGIS with CMAKE	218
I	Building under Windows using MSVC Express	219
I.1	System preparation	219
I.2	Install the libraries archive	219
I.3	Install Visual Studio Express 2005	220
I.4	Install Microsoft Platform SDK2	220

I.5	Edit your vsvars	223
I.6	Environment Variables	224
I.7	Building Qt4.3.2	225
I.7.1	Compile Qt	225
I.7.2	Configure Visual C++ to use Qt	226
I.8	Install Python	227
I.9	Install SIP	227
I.10	Install PyQt4	227
I.11	Install CMake	228
I.12	Install Subversion	228
I.13	Initial SVN Check out	228
I.14	Create Makefiles using cmakesetup.exe	229
I.15	Running and packaging	229
J	QGIS Coding Standards	230
J.1	Classes	230
J.1.1	Names	230
J.1.2	Members	230
J.1.3	Accessor Functions	231
J.1.4	Functions	231
J.2	Qt Designer	231
J.2.1	Generated Classes	231
J.2.2	Dialogs	231
J.3	C++ Files	232
J.3.1	Names	232
J.3.2	Standard Header and License	232
J.3.3	CVS Keyword	232
J.4	Variable Names	233
J.5	Enumerated Types	233
J.6	Global Constants	233
J.7	Editing	233
J.7.1	Tabs	233
J.7.2	Indentation	234
J.7.3	Braces	234
J.8	API Compatibility	234
J.9	Coding Style	235
J.9.1	Where-ever Possible Generalize Code	235
J.9.2	Prefer Having Constants First in Predicates	235
J.9.3	Whitespace Can Be Your Friend	235
J.9.4	Add Trailing Identifying Comments	236
J.9.5	Use Braces Even for Single Line Statements	236
J.9.6	Book recommendations	237

K SVN Access	237
K.1 Accessing the Repository	237
K.2 Anonymous Access	237
K.3 QGIS documentation sources	238
K.4 Documentation	238
K.5 Development in branches	238
K.5.1 Purpose	238
K.5.2 Procedure	239
K.5.3 Creating a branch	239
K.5.4 Merge regularly from trunk to branch	239
K.6 Submitting Patches	240
K.6.1 Patch file naming	240
K.6.2 Create your patch in the top level QGIS source dir	240
K.6.3 Including non version controlled files in your patch	241
K.6.4 Getting your patch noticed	241
K.6.5 Due Diligence	241
K.7 Obtaining SVN Write Access	241
K.7.1 Procedure once you have access	241
L Unit Testing	243
L.1 The QGIS testing framework - an overview	243
L.2 Creating a unit test	244
L.3 Adding your unit test to CMakeLists.txt	250
L.4 Building your unit test	252
L.5 Run your tests	252
M HIG (Human Interface Guidelines)	254
N GNU General Public License	255
N.1 Quantum GIS Qt exception for GPL	260
Cited literature	261

List of Figures

1	A Simple QGIS Session 	11
2	QGIS GUI with Alaska sample data 	14
3	Measure tools in action 	23
4	Open an OGR Supported Vector Layer Dialog 	28
5	QGIS with Shapefile of Alaska loaded 	29
6	Vector Layer Properties Dialog 	36
7	Symbolizing-options 	37
8	Select feature and choose action 	43
9	Edit snapping options on a layer basis 	45
10	Enter Attribute Values Dialog after digitizing a new vector feature 	49
11	Creating a New Vector Dialog 	54
12	Query Builder 	55
13	Raster Layers Properties Dialog 	58
14	Dialog for adding a WMS server, showing its available layers 	65
15	Adding a WFS layer 	70
16	CRS tab in the QGIS Options Dialog 	72
17	Projection Dialog 	73
18	Custom CRS Dialog 	75
19	GRASS data in the alaska LOCATION (adapted from Neteler & Mitasova 2008 (2))	78
20	Creating a new GRASS LOCATION or a new MAPSET in QGIS 	79
21	GRASS Digitizing Toolbar 	83
22	GRASS Digitizing Category Tab 	85
23	GRASS Digitizing Settings Tab 	85
24	GRASS Digitizing Symbolog Tab 	86
25	GRASS Digitizing Table Tab 	86
26	GRASS Toolbox and searchable Modules List 	88
27	GRASS Toolbox Module Dialogs 	88
28	GRASS LOCATION browser 	90
29	Print Composer 	93
30	Print Composer map item tab content 	94
31	Customize print composer label and images 	95
32	Customize print composer legend and scalebar 	96
33	Print Composer with map view, legend, scalebar, and text added 	97
34	Plugin Manager 	99
35	Installing external python plugins 	100
36	Coordinate Capture Plugin 	103
37	Copyright Label Plugin 	104
38	North Arrow Plugin 	105
39	Scale Bar Plugin 	106
40	Delimited Text Dialog 	108

41	Dxf2Shape Converter Plugin 	109
42	Select an image to georeference 	110
43	Arrange plugin window with the qgis map canvas 	111
44	Add points to the raster image 	112
45	Georeferenced map with overlaid roads from spearfish60 location 	113
46	Quick Print Dialog 	114
47	Quick Print result as DIN A4 PDF 	114
48	The <i>GPS Tools</i> dialog window 	116
49	File selection dialog for the import tool 	117
50	The download tool 	118
51	Create a graticule layer 	120
52	Interpolation Plugin 	121
53	Interpolation of elevp data using IDW method 	122
54	Arrange raster and vector layers for QGIS project file 	123
55	Export to MapServer Dialog 	124
56	Test PNG created by shp2img with all MapServer Export layers 	126
57	OGR Layer Converter Plugin 	127
58	Simple C++ Application X	157
59	QMainWindow application with a menu, toolbar and canvas area X	160

List of Tables

1	PostGIS Connection Parameters	31
2	WMS Connection Parameters	64
3	Example Public WMS URLs	65
4	GRASS Digitizing Tools	84
5	Print Composer Tools	92
6	QGIS Core Plugins	102
7	Current moderated external QGIS Plugins	128
8	GRASS Toolbox: Data import modules	175
9	GRASS Toolbox: Data export modules	176
10	GRASS Toolbox: Data type conversion modules	176
11	GRASS Toolbox: Region and projection configuration modules	177
12	GRASS Toolbox: Develop raster map modules	178
13	GRASS Toolbox: Raster color management modules	178
14	GRASS Toolbox: Spatial raster analysis modules	179
15	GRASS Toolbox: Surface management modules	180
16	GRASS Toolbox: Change raster category values and labels modules	180
17	GRASS Toolbox: Hydrologic modelling modules	181
18	GRASS Toolbox: Reports and statistic analysis modules	181
19	GRASS Toolbox: Develop vector map modules	182
20	GRASS Toolbox: Database connection modules	183
21	GRASS Toolbox: Change vector field modules	183
22	GRASS Toolbox: Working with vector points modules	183
23	GRASS Toolbox: Spatial vector and network analysis modules	184
24	GRASS Toolbox: Vector update by other maps modules	184
25	GRASS Toolbox: Vector report and statistic modules	184
26	GRASS Toolbox: Imagery analysis modules	185
27	GRASS Toolbox: Database modules	186
28	GRASS Toolbox: 3D Visualization	187
29	GRASS Toolbox: Reference Manual	187

QGIS Tips

1	UP-TO-DATE DOCUMENTATION	1
2	EXAMPLE USING COMMAND LINE ARGUMENTS	13
3	RESTORING TOOLBARS	17
4	ZOOMING THE MAP WITH THE MOUSE WHEEL	19
5	PANNING THE MAP WITH THE ARROW KEYS AND SPACE BAR	19
6	CALCULATING THE CORRECT SCALE OF YOUR MAP CANVAS	20
7	LAYER COLORS	28
8	QGIS USER SETTINGS AND SECURITY	31
9	POSTGIS LAYERS	32
10	EXPORTING DATASETS FROM POSTGIS	33
11	IMPORTING SHAPEFILES CONTAINING POSTGRESQL RESERVED WORDS	33
12	DATA INTEGRITY	47
13	MANIPULATING ATTRIBUTE DATA	47
14	SAVE REGULARLY	48
15	CONCURRENT EDITS	48
16	ZOOM IN BEFORE EDITING	49
17	VERTEX MARKERS	49
18	ATTRIBUTE VALUE TYPES	50
19	CONGRUENCY OF PASTED FEATURES	52
20	FEATURE DELETION SUPPORT	52
21	CHANGING THE LAYER DEFINITION	55
22	VIEWING A SINGLE BAND OF A MULTIBAND RASTER	59
23	GATHERING RASTER STATISTICS	62
24	ON WMS SERVER URLS	65
25	IMAGE ENCODING	66
26	WMS LAYER ORDERING	66
27	WMS LAYER TRANSPARENCY	66
28	WMS PROJECTIONS	67
29	ACCESSING SECURED OGC-LAYERS	69
30	FINDING WMS AND WFS SERVERS	70
31	PROJECT PROPERTIES DIALOG	74
32	GRASS DATA LOADING	77
33	LEARNING THE GRASS VECTOR MODEL	82
34	CREATING AN ATTRIBUTE TABLE FOR A NEW GRASS VECTOR LAYER	83
35	DIGITIZING POLYGONES IN GRASS	83
36	CREATING AN ADDITIONAL GRASS 'LAYER' WITH QGIS	84
37	GRASS EDIT PERMISSIONS	86
38	DISPLAY RESULTS IMMEDIATELY	89
39	SAVING A PRINT COMPOSER LAYOUT	95
40	CRASHING PLUGINS	98

41	PLUGINS SETTINGS SAVED TO PROJECT	102
42	CHOOSING THE TRANSFORMATION TYPE	112
43	ADD MORE EXTERNAL PLUGINS	128
44	TWO QGIS PYTHON PLUGIN FOLDERS	149
45	DOCUMENTATION FOR PYQGIS	168

1 Forward

Welcome to the wonderful world of Geographical Information Systems (GIS)! Quantum GIS (QGIS) is an Open Source Geographic Information System. The project was born in May of 2002 and was established as a project on SourceForge in June of the same year. We've worked hard to make GIS software (which is traditionally expensive proprietary software) a viable prospect for anyone with basic access to a Personal Computer. QGIS currently runs on most Unix platforms, Windows, and OS X. QGIS is developed using the Qt toolkit (<http://www.trolltech.com>) and C++. This means that QGIS feels snappy to use and has a pleasing, easy-to-use graphical user interface (GUI).

QGIS aims to be an easy-to-use GIS, providing common functions and features. The initial goal was to provide a GIS data viewer. QGIS has reached the point in its evolution where it is being used by many for their daily GIS data viewing needs. QGIS supports a number of raster and vector data formats, with new format support easily added using the plugin architecture (see Appendix A for a full list of currently supported data formats).

QGIS is released under the GNU General Public License (GPL). Developing QGIS under this license means that you can inspect and modify the source code, and guarantees that you, our happy user, will always have access to a GIS program that is free of cost and can be freely modified. You should have received a full copy of the license with your copy of QGIS, and you also can find it in Appendix N.

Tip 1 UP-TO-DATE DOCUMENTATION

The latest version of this document can always be found at <http://download.osgeo.org/qgis/doc/manual/>, or in the documentation area of the QGIS website at <http://qgis.osgeo.org/documentation/>

1.1 Features

QGIS offers many common GIS functionalities provided by core features and plugins. As a short summary they are presented in six categories to gain a first insight.

View data

You can view and overlay vector and raster data in different formats and projections without conversion to an internal or common format. Supported formats include:

- spatially-enabled PostgreSQL tables using PostGIS, vector formats ¹ supported by the installed OGR library, including ESRI shapefiles, MapInfo, SDTS and GML.
- Raster and imagery formats supported by the installed GDAL (Geospatial Data Abstraction Library) library, such as GeoTiff, Erdas Img., ArcInfo Ascii Grid, JPEG, PNG,

¹OGR-supported database formats such as Oracle or MySQL are not yet supported in QGIS.

- GRASS raster and vector data from GRASS databases (location/mapset),
- Online spatial data served as OGC-compliant Web Map Service (WMS) or Web Feature Service (WFS).

Explore data and compose maps

You can compose maps and interactively explore spatial data with a friendly GUI. The many helpful tools available in the GUI include:

- on the fly projection
- map composer
- overview panel
- spatial bookmarks
- identify/select features
- edit/view/search attributes
- feature labeling
- change vector and raster symbology
- add a graticule layer
- decorate your map with a north arrow scale bar and copyright label
- save and restore projects

Create, edit, manage and export data

You can create, edit, manage and export vector maps in several formats. Raster data have to be imported into GRASS to be able to edit and export them into other formats. QGIS offers the following:

- digitizing tools for OGR supported formats and GRASS vector layer
- create and edit shapefiles and GRASS vector layer
- geocode images with the georeferencer plugin
- GPS tools to import and export GPX format, and convert other GPS formats to GPX or down/upload directly to a GPS unit
- create PostGIS layers from shapefiles with the SPIT plugin
- manage vector attribute tables with the table manager plugin

Analyse data

You can perform spatial data analysis on PostgreSQL/PostGIS and other OGR supported formats using the ftools python plugin. QGIS currently offers vector analysis, sampling, geoprocessing, ge-

ometry and database management tools. You can also use the integrated GRASS tools, which include the complete GRASS functionality of more than 300 modules (See Section 9).

Publish maps on the internet

QGIS can be used to export data to a mapfile and to publish them on the internet using a webserver with UMN MapServer installed. QGIS can also be used as a WMS or WFS client, and as WMS server.

Extend QGIS functionality through plugins

QGIS can be adapted to your special needs with the extensible plugin architecture. QGIS provides libraries that can be used to create plugins. You can even create new applications with C++ or Python!

- **Core Plugins**

- Add WFS Layer
- Add Delimited Text Layer
- Coordinate Capture
- Decorations (Copyright Label, North Arrow and Scale bar)
- Georeferencer
- Dxf2Shp Converter
- GPS Tools
- GRASS integration
- Graticules Creator
- Interpolation Plugin
- OGR Layer Converter
- Quick Print
- SPIT Shapefile to PostgreSQL/PostGIS Import Tool
- Mapserver Export
- Python Console
- Python Plugin Installer

- **Python Plugins**



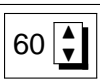


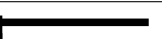
QGIS offers a growing number of external python plugins that are provided by the community. These plugins reside in the the official PyQGIS repository, and can be easily installed using the python plugin installer (See Section 11).

1.2 Conventions

This section describes a collection of uniform styles throughout the manual. The conventions used in this manual are as follows:

GUI Conventions

The GUI convention styles are intended to mimic the appearance of the GUI. In general, the objective is to use the non-hover appearance, so a user can visually scan the GUI to find something that looks like the instruction in the manual.

- Menu Options: **Layer** > **Add a Raster Layer**
or
Settings > **Toolbars** > **Digitizing**
- Tool:  **Add a Raster Layer**
- Button: **Save as Default**
- Dialog Box Title: **Layer Properties**
- Tab: **General**
- Toolbox Item:  **nviz - Open 3D-View in NVIZ**
- Checkbox: **Render**
- Radio Button: **Postgis SRID** **EPSG ID**
- Select a Number: Hue 
- Select a String: Outline style 
- Browse for a File:
- Select a Color: Outline color 
- Slider: Transparency 0% 
- Input Text: Display Name

A shadow indicates a clickable GUI component.

Text or Keyboard Conventions



The manual also includes styles related to text, keyboard commands and coding to indicate different entities, such as classes, or methods. They don't correspond to any actual appearance.



- Hyperlinks: <http://qgis.org>
- Single Keystroke: press **p**
- Keystroke Combinations: press **Ctrl+B**, meaning press and hold the Ctrl key and then press the B key.
- Name of a File: `lakes.shp`
- Name of a Class: **NewLayer**
- Method: `classFactory`
- Server: `myhost.de`
- User Text: `qgis --help`

Code is indicated by a fixed-width font:


```
PROJCS["NAD_1927_Albers",  
      GEOGCS["GCS_North_American_1927",
```


Platform-specific instructions

GUI sequences and small amounts of text can be formatted inline: Click {  File **X** QGIS} > Quit to close QGIS. This indicates that on Linux, Unix and Windows platforms, click the File menu option first, then Quit from the dropdown menu, while on Macintosh OSX platforms, click the QGIS menu option first, then Quit from the dropdown menu. Larger amounts of text may be formatted as a list:

-  do this;
-  do that;
- **X** do something else.

or as paragraphs.

 **X** Do this and this and this. Then do this and this and this and this and this and this and this and this and this and this.

 Do that. Then do that and that and that and that and that and that and that and that and that and that and that and that and that and that and that and that and that and that and that.

Screenshots that appear throughout the user guide have been created on different platforms; the platform is indicated by the platform-specific icons at the end of the figure caption.

2 Introduction To GIS

A Geographical Information System (GIS)⁽¹⁾² is a collection of software that allows you to create, visualize, query and analyze geospatial data. Geospatial data refers to information about the geographic location of an entity. This often involves the use of a geographic coordinate, like a latitude or longitude value. Spatial data is another commonly used term, as are: geographic data, GIS data, map data, location data, coordinate data and spatial geometry data.

Applications using geospatial data perform a variety of functions. Map production is the most easily understood function of geospatial applications. Mapping programs take geospatial data and render it in a form that is viewable, usually on a computer screen or printed page. Applications can present static maps (a simple image) or dynamic maps that are customised by the person viewing the map through a desktop program or a web page.

Many people mistakenly assume that geospatial applications just produce maps, but geospatial data analysis is another primary function of geospatial applications. Some typical types of analysis include computing:

1. distances between geographic locations
2. the amount of area (e.g., square meters) within a certain geographic region
3. what geographic features overlap other features
4. the amount of overlap between features
5. the number of locations within a certain distance of another
6. and so on...

These may seem simplistic, but can be applied in all sorts of ways across many disciplines. The results of analysis may be shown on a map, but are often tabulated into a report to support management decisions.

The recent phenomena of location-based services promises to introduce all sorts of other features, but many will be based on a combination of maps and analysis. For example, you have a cell phone that tracks your geographic location. If you have the right software, your phone can tell you what kind of restaurants are within walking distance. While this is a novel application of geospatial technology, it is essentially doing geospatial data analysis and listing the results for you.

2.1 Why is all this so new?

Well, it's not. There are many new hardware devices that are enabling mobile geospatial services. Many open source geospatial applications are also available, but the existence of geospatially fo-

²This chapter is by Tyler Mitchell (<http://www.oreillynet.com/pub/wlg/7053>) and used under the Creative Commons License. Tyler is the author of *Web Mapping Illustrated*, published by O'Reilly, 2005.

cused hardware and software is nothing new. Global positioning system (GPS) receivers are becoming commonplace, but have been used in various industries for more than a decade. Likewise, desktop mapping and analysis tools have also been a major commercial market, primarily focused on industries such as natural resource management.

What is new is how the latest hardware and software is being applied and who is applying it. Traditional users of mapping and analysis tools were highly trained GIS Analysts or digital mapping technicians trained to use CAD-like tools. Now, the processing capabilities of home PCs and open source software (OSS) packages have enabled an army of hobbyists, professionals, web developers, etc. to interact with geospatial data. The learning curve has come down. The costs have come down. The amount of geospatial technology saturation has increased.

How is geospatial data stored? In a nutshell, there are two types of geospatial data in widespread use today. This is in addition to traditional tabular data that is also widely used by geospatial applications.

2.1.1 Raster Data

One type of geospatial data is called raster data or simply "a raster". The most easily recognised form of raster data is digital satellite imagery or air photos. Elevation shading or digital elevation models are also typically represented as raster data. Any type of map feature can be represented as raster data, but there are limitations.

A raster is a regular grid made up of cells, or in the case of imagery, pixels. They have a fixed number of rows and columns. Each cell has a numeric value and has a certain geographic size (e.g. 30x30 meters in size).

Multiple overlapping rasters are used to represent images using more than one colour value (i.e. one raster for each set of red, green and blue values is combined to create a colour image). Satellite imagery also represents data in multiple "bands". Each band is essentially a separate, spatially overlapping raster, where each band holds values of certain wavelengths of light. As you can imagine, a large raster takes up more file space. A raster with smaller cells can provide more detail, but takes up more file space. The trick is finding the right balance between cell size for storage purposes and cell size for analytical or mapping purposes.

2.1.2 Vector Data

Vector data is also used in geospatial applications. If you stayed awake during trigonometry and coordinate geometry classes, you will already be familiar with some of the qualities of vector data. In its simplest sense, vectors are a way of describing a location by using a set of coordinates. Each coordinate refers to a geographic location using a system of x and y values.

This can be thought of in reference to a Cartesian plane - you know, the diagrams from school

that showed an x and y-axis. You might have used them to chart declining retirement savings or increasing compound mortgage interest, but the concepts are essential to geospatial data analysis and mapping.

There are various ways of representing these geographic coordinates depending on your purpose. This is a whole area of study for another day - map projections.

Vector data takes on three forms, each progressively more complex and building on the former.




1. Points - A single coordinate (x y) represents a discrete geographic location
2. Lines - Multiple coordinates (x1 y1, x2 y2, x3 y4, ... xn yn) strung together in a certain order, like drawing a line from Point (x1 y1) to Point (x2 y2) and so on. These parts between each point are considered line segments. They have a length and the line can be said to have a direction based on the order of the points. Technically, a line is a single pair of coordinates connected together, whereas a line string is multiple lines connected together.
3. Polygons - When lines are strung together by more than two points, with the last point being at the same location as the first, we call this a polygon. A triangle, circle, rectangle, etc. are all polygons. The key feature of polygons is that there is a fixed area within them.

3 Getting Started

This chapter gives a quick overview of installing QGIS, some sample data from the QGIS web page and running a first and simple session visualizing raster and vector layers.


3.1 Installation

Installation of QGIS is very simple. Standard installer packages are available for MS Windows and Mac OS X. For many flavors of GNU/Linux binary packages (rpm and deb) or software repositories to add to your installation manager are provided. Get the latest information on binary packages at the QGIS website at <http://qgis.osgeo.org/download/>.



If you need to build QGIS from source, this is documented in Appendix D for MS Windows , Appendix E for Mac OS X  and Appendix F for GNU/Linux . The Installation instructions are distributed with the QGIS source code and also available at <http://qgis.osgeo.org>.

3.2 Sample Data

The user guide contains examples based on the QGIS sample dataset.

 The Windows installer has an option to download the QGIS sample dataset. If checked, the data will be downloaded to your My Documents folder and placed in a folder called GIS Database. You may use Windows Explorer to move this folder to any convenient location. If you did not select the checkbox to install the sample dataset during the initial QGIS installation, you can either

- use GIS data that you already have;
- download the sample data from the QGIS website <http://qgis.osgeo.org/download>; or
- uninstall QGIS and reinstall with the data download option checked.

  For GNU/Linux and Mac OS X there are not yet dataset installation packages available as rpm, deb or dmg. To use the sample dataset download the file `qgis_sample_data` as ZIP or TAR archive from <http://download.osgeo.org/qgis/data/> and unzip or untar the archive on your system. The Alaska dataset includes all GIS data that are used as examples and screenshots in the user guide, and also includes a small GRASS database. The projection for the QGIS sample dataset is Alaska Albers Equal Area with unit feet. The EPSG code is 2964.

```
PROJCS["Albers Equal Area",  
    GEOGCS["NAD27",  
        DATUM["North_American_Datum_1927",
```



```
SPHEROID["Clarke 1866",6378206.4,294.978698213898,  
    AUTHORITY["EPSG","7008"]],  
TOWGS84[-3,142,183,0,0,0,0],  
AUTHORITY["EPSG","6267"]],  
PRIMEM["Greenwich",0,  
    AUTHORITY["EPSG","8901"]],  
UNIT["degree",0.0174532925199433,  
    AUTHORITY["EPSG","9108"]],  
AUTHORITY["EPSG","4267"]],  
PROJECTION["Albers_Conic_Equal_Area"],  
PARAMETER["standard_parallel_1",55],  
PARAMETER["standard_parallel_2",65],  
PARAMETER["latitude_of_center",50],  
PARAMETER["longitude_of_center",-154],  
PARAMETER["false_easting",0],  
PARAMETER["false_northing",0],  
UNIT["us_survey_feet",0.3048006096012192]]
```

If you intend to use QGIS as graphical frontend for GRASS, you can find a selection of sample locations (e.g. Spearfish or South Dakota) at the official GRASS GIS-website <http://grass.osgeo.org/download/data.php>.

3.3 Sample Session

Now that you have QGIS installed and a sample dataset available, we would like to demonstrate a short and simple QGIS sample session. We will visualize a raster and a vector layer. We will use the landcover raster layer `qgis_sample_data/raster/landcover.img` and the lakes vector layer `qgis_sample_data/gml/lakes.gml`.

start QGIS

-  Start QGIS by typing: `qgis` at a command prompt.
-  Start QGIS using the Start menu or desktop shortcut, or double click on a QGIS project file.
- **X** double click the icon in your Applications folder.

Load raster and vector layers from the sample dataset

1. Click on the  **Load Raster** icon.


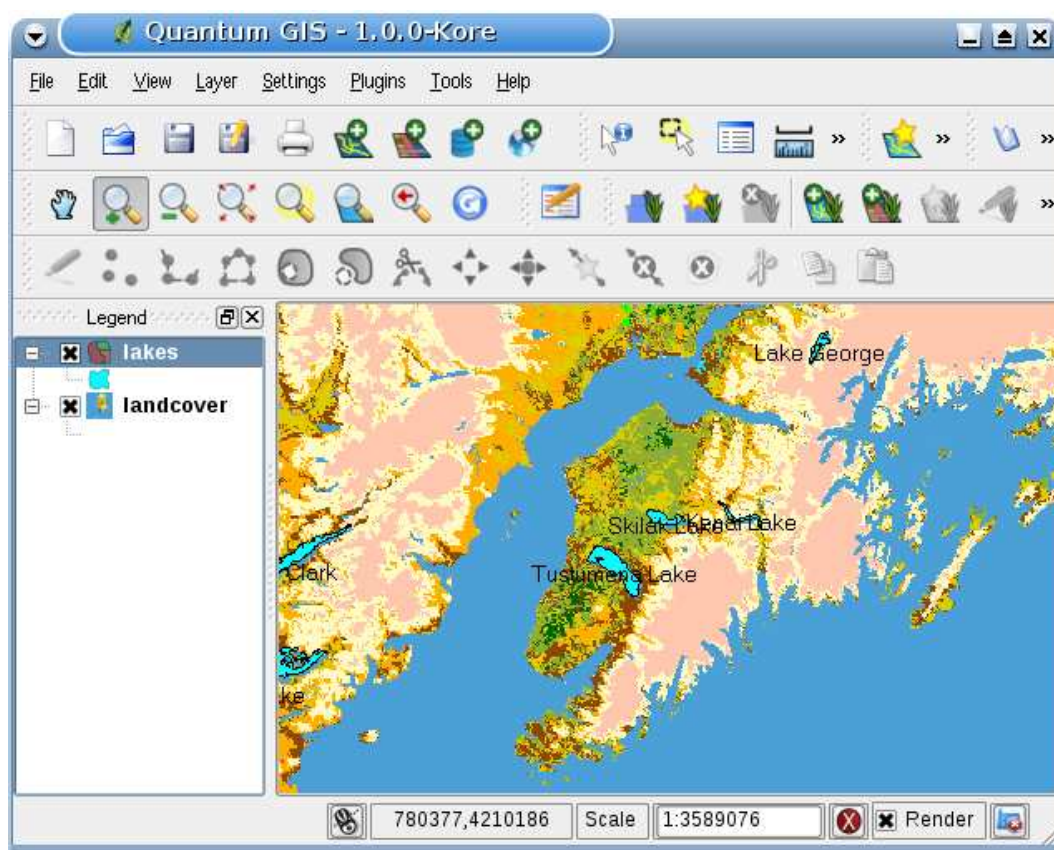
2. Browse to the folder `qgis_sample_data/raster/`, select the ERDAS Img file `landcover.img` and click **Open**.
3. Now click on the  **Load Vector** icon.
4. browse to the folder `qgis_sample_data/gml/`, select the GML file `lakes.gml` and click **Open**.
5. Zoom in a bit to your favorite area with some lakes.
6. Double click the `lakes` layer in the map legend to open the **Layer Properties** dialog.
7. Click on the **Symbology** tab and select a blue as fill color.
8. Click on the **Labels** tab and check the **Display labels** checkbox to enable labeling.
9. Click **Apply**.

Figure 1: A Simple QGIS Session 





You can see how easy it is to visualize raster and vector layers in QGIS. Let's move on to the sections that follow to learn more about the available functionality, features and settings and how to use them.

4 Features at a Glance

After a first and simple sample session in Section 3 we now want to give you a more detailed overview of the features of QGIS. Most features presented in the following chapters will be explained and described in own sections later in the manual.


4.1 Starting and Stopping QGIS

In Section 3.3 you already learned how to start QGIS. We will repeat this here and you will see that QGIS also provides further command line options.

-  assuming that QGIS is installed in the PATH, you can start QGIS by typing: `qgis` at a command prompt or by double clicking on the QGIS application link (or shortcut) on the desktop.
-  start QGIS using the Start menu or desktop shortcut, or double click on a QGIS project file.
- **X** double click the icon in your Applications folder.

To stop QGIS, click the menu options   File **X** QGIS > Quit, or use the shortcut Ctrl+Q.

4.1.1 Command Line Options

 QGIS supports a number of options when started from the command line. To get a list of the options, enter `qgis --help` on the command line. The usage statement for QGIS is:

```
qgis --help
Quantum GIS - 1.0.0 'Kore'
Quantum GIS (QGIS) is a viewer for spatial data sets, including
raster and vector data.
Usage: qgis [options] [FILES]
options:
    [--snapshot filename]  emit snapshot of loaded datasets to given file
    [--lang language]      use language for interface text
    [--project projectfile] load the given QGIS project
    [--extent xmin,ymin,xmax,ymax] set initial map extent
    [--help]                this text
```

FILES:

Files specified on the command line can include rasters, vectors, and QGIS project files (.qgs):

1. Rasters - Supported formats include GeoTiff, DEM and others supported by GDAL

2. Vectors - Supported formats include ESRI Shapefiles and others supported by OGR and PostgreSQL layers using the PostGIS extension

Tip 2 EXAMPLE USING COMMAND LINE ARGUMENTS

You can start QGIS by specifying one or more data files on the command line. For example, assuming you are in the `qgis_sample_data` directory, you could start QGIS with a vector layer and a raster file set to load on startup using the following command: `qgis ./raster/landcover.img ./gml/lakes.gml`

Command line option `--snapshot`

This option allows you to create a snapshot in PNG format from the current view. This comes in handy when you have a lot of projects and want to generate snapshots from your data.

Currently it generates a PNG-file with 800x600 pixels. A filename can be added after `--snapshot`.

Command line option `--lang`

Based on your locale QGIS, selects the correct localization. If you would like to change your language, you can specify a language code. For example: `--lang=it` starts QGIS in italian localization. A list of currently supported languages with language code is provided at <http://wiki.qgis.org/qgiswiki/TranslatorsCorner>

Command line option `--project`

Starting QGIS with an existing project file is also possible. Just add the command line option `-project` followed by your project name and QGIS will open with all layers loaded described in the given file.

Command line option `--extent`


To start with a specific map extent use this option. You need to add the bounding box of your extent in the following order separated by a comma:

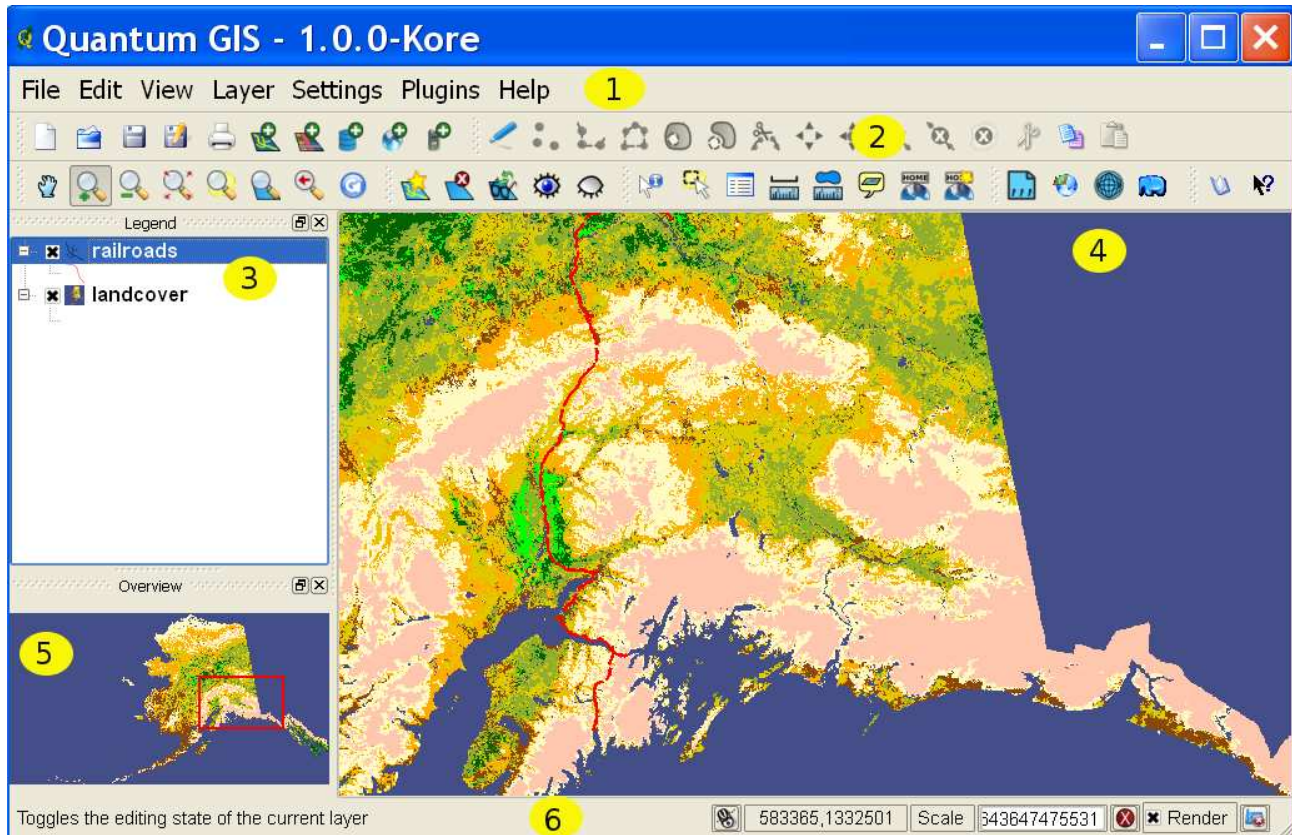
```
--extent xmin,ymin,xmax,ymax
```

4.2 QGIS GUI

When QGIS starts, you are presented with the GUI as shown below (the numbers 1 through 6 in yellow ovals refer to the six major areas of the interface as discussed below):

Note: Your window decorations (title bar, etc.) may appear different depending on your operating system and window manager.

Figure 2: QGIS GUI with Alaska sample data 



The QGIS GUI is divided into six areas:


- | | |
|---------------|-----------------|
| 1. Menu Bar | 4. Map View |
| 2. Tool Bar | 5. Map Overview |
| 3. Map Legend | 6. Status Bar |

These six components of the QGIS interface are described in more detail in the following sections.

4.2.1 Menu Bar

The menu bar provides access to various QGIS features using a standard hierarchical menu. The top-level menus and a summary of some of the menu options are listed below, together with the icons of the corresponding tools as they appear on the toolbar, as well as keyboard shortcuts. Although most menu options have a corresponding tool and vice-versa, the menus are not organized quite like the toolbars. The toolbar containing the tool is listed after each menu option as a checkbox entry. For more information about tools and toolbars, see Section 4.2.2.

Menu Option	Shortcut	Reference	Toolbar
• File			
New Project	Ctrl+N	see Section 4.5	
Open Project	Ctrl+O	see Section 4.5	
Open Recent Projects		see Section 4.5	
Save Project	Ctrl+S	see Section 4.5	
Save Project As	Ctrl+Shift+S	see Section 4.5	
Save as Image		see Section 4.6	
Print Composer	Ctrl+P	see Section 10	
Exit	Ctrl+Q		
• Edit			
Cut Features	Ctrl+X	see Section 5.4.3	
Copy Features	Ctrl+C	see Section 5.4.3	
Paste Features	Ctrl+V	see Section 5.4.3	
Capture Point	.	see Section 5.4.3	
Capture Line	/	see Section 5.4.3	
Capture Polygon	Ctrl+/	see Section 5.4.3	
And Other Edit Menu Items		see Section 5.4.3	
• View			
Pan Map			
Zoom In	Ctrl++		
Zoom Out	Ctrl+-		
Select Features			
Identify Features	I		
Measure Line	M		
Measure Area	J		
Zoom Full	F		
Zoom To Layer			
Zoom To Selection	Ctrl+J		
Zoom Last			
Zoom Actual Size			

 Map Tips

 New Bookmark

Ctrl+B

see Section 4.8

 Show Bookmarks

B

see Section 4.8


 Refresh

Ctrl+R


 Attributes

 Attributes

 Attributes


 Map Navigation


- Layer

 New Vector Layer

N

see Section 5.4.4

 Manage Layers

 Add a Vector Layer

V

see Section 5

 File

 Add a Raster Layer

R

see Section 6


 File

 Add a PostGIS Layer

D

see Section 5.2

 File

 Add a WMS Layer


W

see Section 7.2

 File

 Open Attribute Table


 Attributes

 Toggle editing


 Digitizing

Save As Shapefile


Save Selection As Shapefile

 Remove Layer


Ctrl+D


 Manage Layers

Properties

 Add to Overview

O

 Manage Layers

 Add All To Overview


+


 Remove All From Overview

-


 Hide All Layers

H

 Manage Layers

 Show All Layers

S


 Manage Layers

- Settings

Panels


Toolbars

Toggle Fullscreen Mode


 Project Properties

P

see Section 4.5


 Custom CRS


see Section 8.4

 Options

see Section 4.7

- Plugins - (Futher menu items are added by plugins as they are loaded.)

 Plugin Manager

see Section 11.1 

- **Help**
 - Help Contents (F1)
 - QGIS Home Page (Ctrl+H)
 - Check QGIS Version
 - About
- ☒ **Help**

4.2.2 Toolbars

The toolbars provide access to most of the same functions as the menus, plus additional tools for interacting with the map. Each toolbar item has popup help available. Hold your mouse over the item and a short description of the tool's purpose will be displayed.

Every menubar can be moved around according to your needs. Additionally every menubar can be switched off using your right mouse button context menu holding the mouse over the toolbars.

Tip 3 RESTORING TOOLBARS

If you have accidentally hidden all your toolbars, you can get them back by choosing menu option **Settings** > **Toolbars**.

4.2.3 Map Legend

The map legend area is used to set the visibility and z-ordering of layers. Z-ordering means that layers listed nearer the top of the legend are drawn over layers listed lower down in the legend. The checkbox in each legend entry can be used to show or hide the layer.

Layers can be grouped in the legend window by adding a layer group and dragging layers into the group. To do so, move the mouse pointer to the legend window, right click, choose **Add group**. A new folder appears. Now drag the layers onto to the folder symbol. It is then possible to toggle the visibility of all the layers in the group with one click. To bring layers out of a group, move the mouse pointer to the layer symbol, right click, and choose **Make to toplevel item**. To give the folder a new name, choose **Rename** in the right click menu of the group.

The content of the right mouse button context menu depends on whether the loaded legend item you hold your mouse over is a raster or a vector layer. For GRASS vector layers the **toggle editing** is not available. See section 9.7 for information on editing GRASS vector layers.

- **Right mouse button menu for raster layers**
 - **Zoom to layer extent**

- Zoom to best scale (100%)
- Show in overview
- Remove
- Properties
- Rename
- Add Group
- Expand all
- Collapse all
- Show file groups

- **Right mouse button menu for vector layers**

- Zoom to layer extent
- Show in overview
- Remove
- Open attribute table
- Toggle editing (not available for GRASS layers)
- Save as shapefile
- Save selection as shapefile
- Properties
- Make to toplevel item
- Rename
- Add Group
- Expand all
- Collapse all
- Show file groups

- **Right mouse button menu for layer groups**

- Remove
- Rename
- Add Group
- Expand all
- Collapse all
- Show file groups

If several vector data sources have the same vector type and the same attributes, their symbolisations may be grouped. This means that if the symbolisation of one data source is changed, the others automatically have the new symbolisation as well. To group symbologies, open the right click menu in the legend window and choose **Show file groups**. The file groups of the layers appear. It is now possible to drag a file from one file group into another one. If this is done, the symbologies are grouped. Note that QGIS only permits the drag if the two layers are able to share symbology (same vector geometry type and same attributes).

4.2.4 Map View

This is the 'business end' of QGIS - maps are displayed in this area! The map displayed in this window will depend on the vector and raster layers you have chosen to load (see sections that follow for more information on how to load layers). The map view can be panned (shifting the focus of the map display to another region) and zoomed in and out. Various other operations can be performed on the map as described in the toolbar description above. The map view and the legend are tightly bound to each other - the maps in view reflect changes you make in the legend area.

Tip 4 ZOOMING THE MAP WITH THE MOUSE WHEEL

You can use the mouse wheel to zoom in and out on the map. Place the mouse cursor inside the map area and roll the wheel forward (away from you) to zoom in and backwards (towards you) to zoom out. The mouse cursor position is the center where the zoom occurs. You can customize the behavior of the mouse wheel zoom using the **Map tools** tab under the **Settings** > **Options** menu.

Tip 5 PANNING THE MAP WITH THE ARROW KEYS AND SPACE BAR

You can use the arrow keys to pan in the map. Place the mouse cursor inside the map area and click on the right arrow key to pan East, left arrow key to pan West, up arrow key to pan North and down arrow key to pan South. You can also pan the map using the space bar: just move the mouse while holding down space bar.

4.2.5 Map Overview

The map overview area provides a full extent view of layers added to it. Within the view is a rectangle showing the current map extent. This allows you to quickly determine which area of the map you are currently viewing. Note that labels are not rendered to the map overview even if the layers in the map overview have been set up for labeling. You can add a single layer to the overview by right-clicking on it in the legend and select **Show in overview**. You can also add layers to, or remove all layers from the overview using the Overview tools on the toolbar.

If you click and drag the red rectangle in the overview that shows your current extent, the main map view will update accordingly.

4.2.6 Status Bar


The status bar shows you your current position in map coordinates (e.g. meters or decimal degrees) as the mouse pointer is moved across the map view. To the left of the coordinate display in the status bar is a small button that will toggle between showing coordinate position or the view extents of the map view as you pan and zoom in and out.

A progress bar in the status bar shows progress of rendering as each layer is drawn to the map view. In some cases, such as the gathering of statistics in raster layers, the progress bar will be used to show the status of lengthy operations.

If a new plugin or a plugin update is available, you will see a message in the status bar. On the right side of the status bar is a small checkbox which can be used to temporarily prevent layers being rendered to the map view (see Section 4.3 below). At the far right of the status bar is a projector icon. Clicking on this opens the projection properties for the current project.

Tip 6 CALCULATING THE CORRECT SCALE OF YOUR MAP CANVAS

When you start QGIS, degrees is the default unit, and it tells QGIS that any coordinate in your layer is in degrees. To get correct scale values, you can either change this to meter manually in the **General** tab under **Settings** > **Project Properties** or you can select a project Coordinate Reference System (CRS)

clicking on the  **projector** icon in the lower right-hand corner of the statusbar. In the last case, the units are set to what the project projection specifies, e.g. '+units=m'.

4.3 Rendering

By default, QGIS renders all visible layers whenever the map canvas must be refreshed. The events that trigger a refresh of the map canvas include:

- Adding a layer
- Panning or zooming
- Resizing the QGIS window
- Changing the visibility of a layer or layers

QGIS allows you to control the rendering process in a number of ways.

4.3.1 Scale Dependent Rendering

Scale dependent rendering allows you to specify the minimum and maximum scales at which a layer will be visible. To set scale dependency rendering, open the **Properties** dialog by double-clicking

on the layer in the legend. On the **General** tab, set the minimum and maximum scale values and then click on the **Scale dependent visibility** checkbox.

You can determine the scale values by first zooming to the level you want to use and noting the scale value in the QGIS status bar.

4.3.2 Controlling Map Rendering

Map rendering can be controlled in the following ways:

a) Suspending Rendering

To suspend rendering, click the **Render** checkbox in the lower right corner of the statusbar.

When the **Render** box is not checked, QGIS does not redraw the canvas in response to any of the events described in Section 4.3. Examples of when you might want to suspend rendering include:

- Add many layers and symbolize them prior to drawing
- Add one or more large layers and set scale dependency before drawing
- Add one or more large layers and zoom to a specific view before drawing
- Any combination of the above

Checking the **Render** box enables rendering and causes an immediate refresh of the map canvas.

b) Setting Layer Add Option

You can set an option to always load new layers without drawing them. This means the layer will be added to the map, but its visibility checkbox in the legend will be unchecked by default. To set this option, choose menu option **Settings** > **Options** and click on the **Rendering** tab. Uncheck the **By default new layers added to the map should be displayed** checkbox. Any layer added to the map will be off (invisible) by default.

c) Updating the Map Display During Rendering

You can set an option to update the map display as features are drawn. By default, QGIS does not display any features for a layer until the entire layer has been rendered. To update the display as features are read from the datastore, choose menu option **Settings** > **Options** click on the **Rendering** tab. Set the feature count to an appropriate value to update the display during rendering. Setting a value of 0 disables update during drawing (this is the default). Setting a value too low

will result in poor performance as the map canvas is continually updated during the reading of the features. A suggested value to start with is 500.

d) Influence Rendering Quality

To influence the rendering quality of the map you have 3 options. Choose menu option **Settings** > **Options** click on the **Rendering** tab and select or deselect following checkboxes.

- Make lines appear less jagged at the expense of some drawing performance
- Fix problems with incorrectly filled polygons
- Continuously redraw the map when dragging the legend/map divider

4.4 Measuring

Measuring works within projected coordinate systems only (e.g., UTM). If the loaded map is defined with a geographic coordinate system (latitude/longitude), the results from line or area measurements will be incorrect. To fix this you need to set an appropriate map coordinate system (See Section 8).

4.4.1 Measure length and areas



QGIS is also able to measure real distances between given points according to a defined ellipsoid. To configure this, choose menu option **Settings** > **Options**, click on the **Map tools** tab and choose the appropriate ellipsoid. The tool then allows you to click points on the map. Each segment-length shows up in the measure-window and additionally the total length is printed. To stop measuring click your right mouse button.



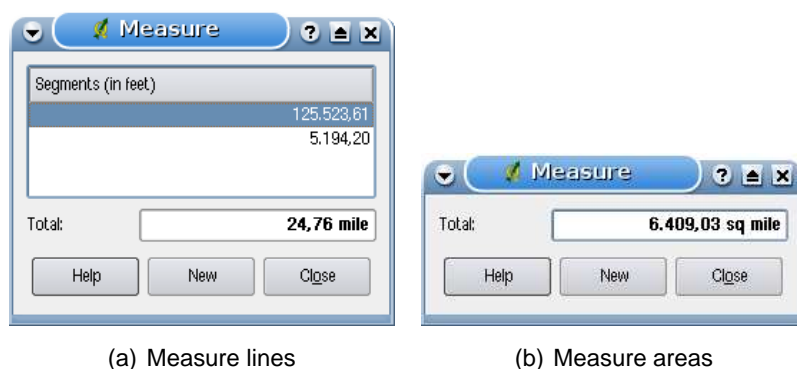
Areas can also be measured. The window shows the accumulated area-size in the measure window

4.5 Projects

The state of your QGIS session is considered a Project. QGIS works on one project at a time. Settings are either considered as being per-project, or as a default for new projects (see Section 4.7). QGIS can save the state of your workspace into a project file using the menu options **File** >

Save Project or **File** > **Save Project As**.

Figure 3: Measure tools in action 🐧



Load saved projects into a QGIS session using **File** > **Open Project** or **File** > **Open Recent Project**. If you wish to clear your session and start fresh, choose **File** > **New Project**. Either of these menu options will prompt you to save the existing project if changes have been made since it was opened or last saved.

The kinds of information saved in a project file include:

- Layers added
- Layer properties, including symbolization
- Projection for the map view
- Last viewed extent

The project file is saved in XML format, so it is possible to edit the file outside QGIS if you know what you are doing. The file format was updated several times compared to earlier QGIS versions. Project files from older QGIS versions may not work properly anymore. To be made aware of this, in the **General** tab under **Settings** > **Options** you can select Warn when opening a project file saved with an older version of QGIS.

4.6 Output

There are several ways to generate output from your QGIS session. We have discussed one already in Section 4.5: saving as a project file. Here is a sampling of other ways to produce output files:

- Menu option **Save as Image** opens a file dialog where you select the name, path and type of image (PNG or JPG format).
- Menu option **Print Composer** opens a dialog where you can layout and print the current map canvas (see Section 10).

4.7 GUI Options



Some basic options for QGIS can be selected using the **Options** dialog. Select the menu option **Settings** > **Options**. The tabs where you can optimize your options are:

General Tab

- Ask to save project changes when required
- Warn when opening a project file saved with an older version of QGIS
- Change Selection and background Color
- Change the icon theme (choose between default, classic, gis and nkids)
- Capitalise layer names in legend
- Display classification attribute names in legend
- Hide splash screen at startup
- Open attribute table in a dock window
- Define attribute table behavior (choose between show all features, show selected features and show features in current canvas)

Rendering Tab

- By default new layers added to the map should be displayed
- Define number of features to draw before updating the display.
- Make lines appear less jagged at the expense of some drawing performance
- Fix problems with incorrectly filled polygons
- Continuously redraw when dragging the legend/map divider

Map tools Tab

- Define Search Radius as a percentage of the map width
- Define Ellipsoid for distance calculations
- Define Rubberband Color for Measure Tools
- Define Mouse wheel action (Zoom, Zoom and recenter, Zoom to mouse cursor, Nothing)

- Define Zoom factor for wheel mouse

Digitizing Tab

- Define Rubberband Color and line width for Digitizing
- Define default snap mode (to vertex, to segment, to vertex and segment)
- Define default snapping tolerance in layer units
- Define search radius for vertex edits in layer units
- Define vertex marker style (Cross or semi transparent circle)

CRS Tab

- Prompt for Coordinate Reference System (CRS)
- Project wide default Coordinate Reference System (CRS) will be used
- Global default Coordinate Reference System (CRS) displayed below will be used
- Select global default Coordinate Reference System (CRS)




Locale Tab

- Overwrite system locale and use defined locale instead
- Information about active system locale

Proxy Tab

- Use proxy for web access and define host, port, user, and password.

You can modify the options according to your needs. Some of the changes may require a restart of QGIS before they will be effective.

-  settings are saved in a texfile: \$HOME/.config/QuantumGIS/qgis.conf
-  you can find your settings in: \$HOME/Library/Preferences/org.qgis.qgis.plist
-  settings are stored in the registry under:

```
\\HKEY\CURRENT\USER\Software\QuantumGIS\qgis
```

4.8 Spatial Bookmarks

Spatial Bookmarks allow you to “bookmark” a geographic location and return to it later.

4.8.1 Creating a Bookmark

To create a bookmark:

1. Zoom or pan to the area of interest.
2. Select the menu option **View** > **New Bookmark** or press **Ctrl-B**.
3. Enter a descriptive name for the bookmark (up to 255 characters).
4. Click **OK** to add the bookmark or **Cancel** to exit without adding the bookmark.

Note that you can have multiple bookmarks with the same name.

4.8.2 Working with Bookmarks

To use or manage bookmarks, select the menu option **View** > **Show Bookmarks**. The **Geospatial Bookmarks** dialog allows you to zoom to or delete a bookmark. You can not edit the bookmark name or coordinates.

4.8.3 Zooming to a Bookmark

From the **Geospatial Bookmarks** dialog, select the desired bookmark by clicking on it, then click **Zoom To**. You can also zoom to a bookmark by double-clicking on it.

4.8.4 Deleting a Bookmark

To delete a bookmark from the **Geospatial Bookmarks** dialog, click on it then click **Delete**. Confirm your choice by clicking **Yes** or cancel the delete by clicking **No**.

5 Working with Vector Data

QGIS supports vector data in a number of formats, including those supported by the OGR library data provider plugin, such as ESRI shapefiles, MapInfo MIF (interchange format) and MapInfo TAB (native format). You find a list of OGR supported vector formats in Appendix A.1.

QGIS also supports PostGIS layers in a PostgreSQL database using the PostgreSQL data provider plugin. Support for additional data types (eg. delimited text) is provided by additional data provider plugins.

This section describes how to work with two common formats: ESRI shapefiles and PostGIS layers. Many of the features available in QGIS work the same regardless of the vector data source. This is by design and includes the identify, select, labeling and attributes functions.

Working with GRASS vector data is described in Section 9.



5.1 ESRI Shapefiles

The standard vector file format used in QGIS is the ESRI Shapefile. It's support is provided by the OGR Simple Feature Library (<http://www.gdal.org/ogr/>) . A shapefile actually consists of a minimum of three files:

- .shp file containing the feature geometries.
- .dbf file containing the attributes in dBase format.
- .shx index file.

Ideally it comes with another file with a .prj suffix, that contains the projection information for the shapefile. There can be more files belonging to a shapefile dataset. To have a closer look at this we recommend the technical specification for the shapefile format, that can be found at <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> .

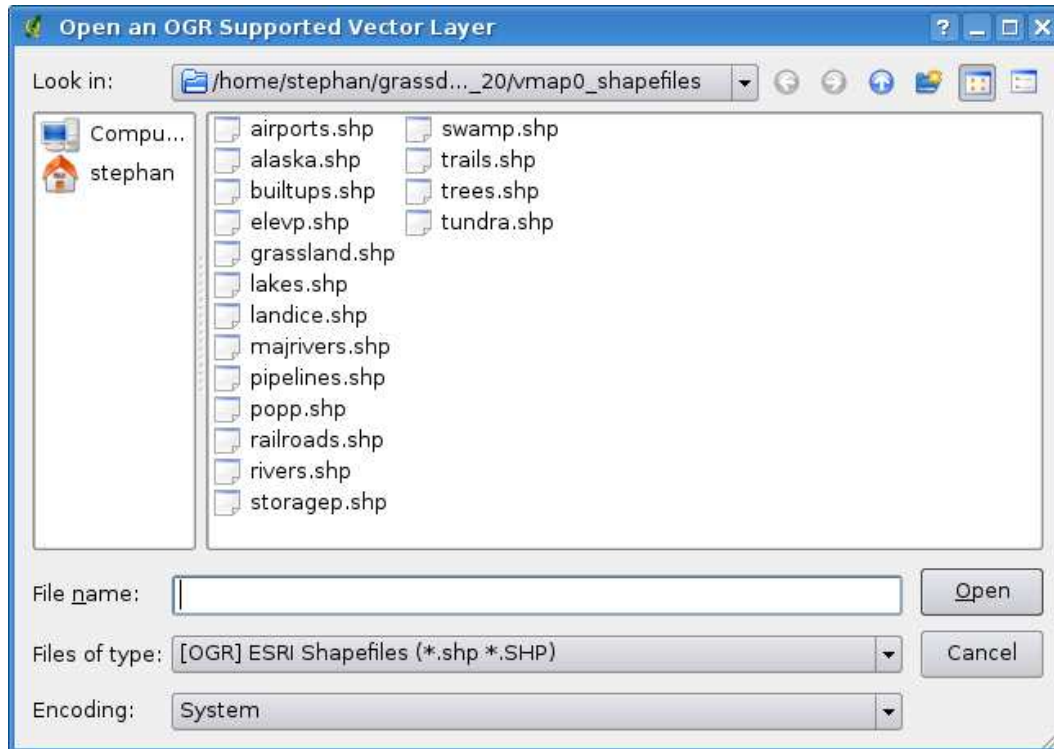
5.1.1 Loading a Shapefile

 To load a shapefile, start QGIS and click on the  **Add a vector layer** toolbar button or simply type . This same tool can be used to load any of the formats supported by the OGR library.

Clicking on the tool brings up a standard open file dialog (see Figure 4) which allows you to navigate the file system and load a shapefile or other supported data source. The selection box allows you to preselect some OGR supported file formats.

You can also select the Encoding type for the shapefile if desired.

Figure 4: Open an OGR Supported Vector Layer Dialog 



Selecting a shapefile from the list and clicking **Open** loads it into QGIS. Figure 5 shows QGIS after loading the `alaska.shp` file.

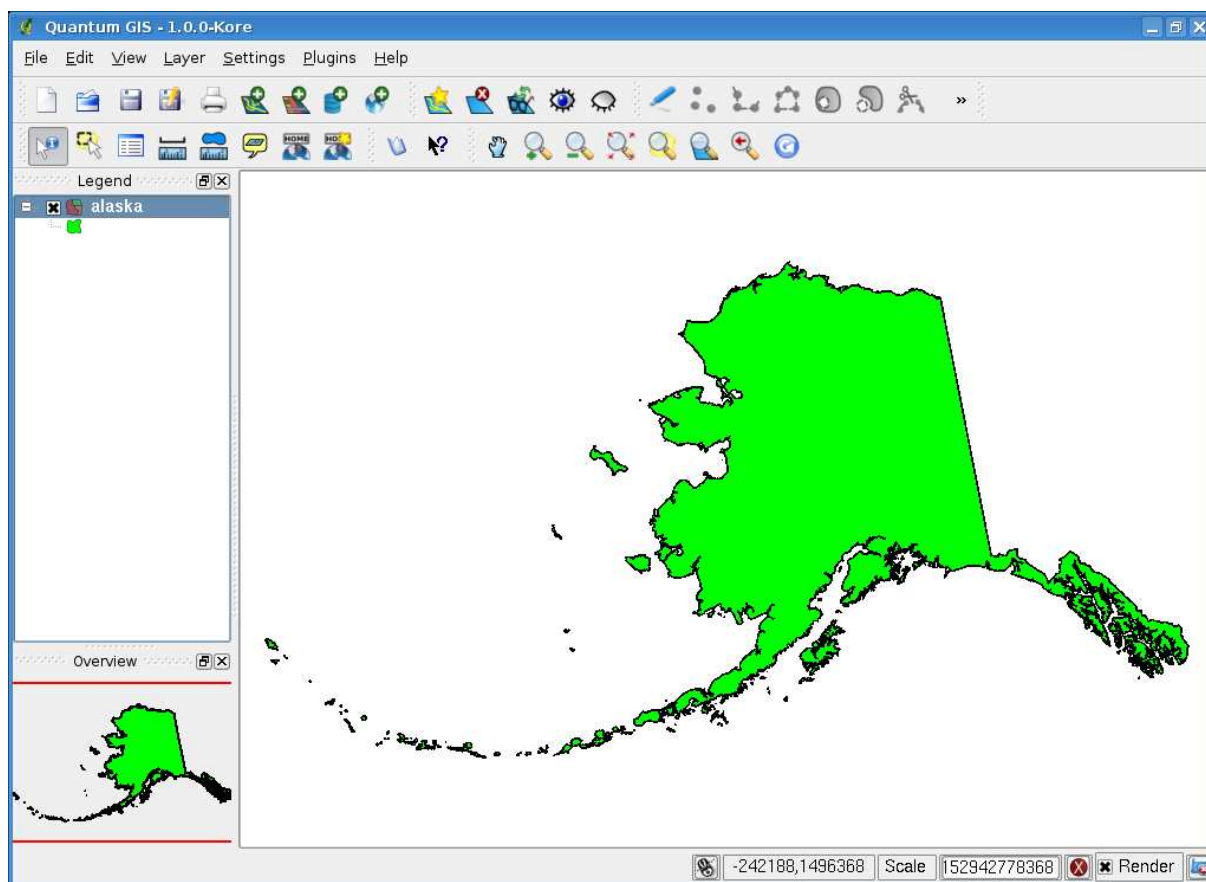
Tip 7 LAYER COLORS

When you add a layer to the map, it is assigned a random color. When adding more than one layer at a time, different colors are assigned to each layer.

Once loaded, you can zoom around the shapefile using the map navigation tools. To change the symbology of a layer, open the **Layer Properties** dialog by double clicking on the layer name or by right-clicking on the name in the legend and choosing **Properties** from the popup menu. See Section 5.3.2 for more information on setting symbology of vector layers.

5.1.2 Improving Performance


To improve the performance of drawing a shapefile, you can create a spatial index. A spatial index will improve the speed of both zooming and panning. Spatial indexes used by QGIS have a `.qix` extension.

Figure 5: QGIS with Shapefile of Alaska loaded 


Use these steps to create the index:

- Load a shapefile.
- Open the **Layer Properties** dialog by double-clicking on the shapefile name in the legend or by right-clicking and choosing **Properties** from the popup menu.
- In the tab **General** click the **Create Spatial Index** button.

5.1.3 Loading a MapInfo Layer

To load a MapInfo layer, click on the  **Add a vector layer** toolbar bar button or type (V), change the file type filter to **Files of Type [OGR] MapInfo (*.mif *.tab *.MIF *.TAB) ▾** and select the layer you want to load.

5.1.4 Loading an ArcInfo Coverage

Loading an ArcInfo coverage is done using the same method as with a shapefiles and MapInfo layers. Click on the  **Add a vector layer** toolbar button or type **(V)** to open the **Open on OGR Supported Vector Layer** dialog and change the file type filter to **Files of Type All files (*.*)**. Navigate to the coverage directory and select one of the following files (if present in your coverage):

- .lab - to load a label layer (polygon labels or standing points).
- .cnt - to load a polygon centroid layer
- .arc - to load an arc (line) layer.
- .pal - to load a polygon layer.

5.2 PostGIS Layers

PostGIS layers are stored in a PostgreSQL database. The advantages of PostGIS are the spatial indexing, filtering and query capabilities it provides. Using PostGIS, vector functions such as select and identify work more accurately than with OGR layers in QGIS.

To use PostGIS layers you must:

- Create a stored connection in QGIS to the PostgreSQL database (if one is not already defined).
- Connect to the database.
- Select the layer to add to the map.
- Optionally provide a SQL `where` clause to define which features to load from the layer.
- Load the layer.

5.2.1 Creating a stored Connection



 The first time you use a PostGIS data source, you must create a connection to the PostgreSQL database that contains the data. Begin by clicking on the  **Add a PostGIS Layer** toolbar button, selecting the **Add a PostGIS Layer...** option from the **Layer** menu or typing **(D)**. The **Add PostGIS Table(s)** dialog will be displayed. To access the connection manager, click on the **New** button to display the **Create a New PostGIS Connection** dialog. The parameters required for a connection are shown in table 1.

Table 1: PostGIS Connection Parameters



Name	A name for this connection. Can be the same as <i>Database</i> .
Host	Name of the database host. This must be a resolvable host name the same as would be used to open a telnet connection or ping the host. If the database is on the same computer as QGIS, simply enter 'localhost' here.
Database	Name of the database.
Port	Port number the PostgreSQL database server listens on. The default port is 5432.
Username	User name used to login to the database.
Password	Password used with <i>Username</i> to connect to the database.

Optional you can activate following checkboxes:


- Save Password
- Only look in the geometry_columns table
- Only look in the 'public' schema

Once all parameters and options are set, you can test the connection by clicking on the **Test Connect** button.


Tip 8 QGIS USER SETTINGS AND SECURITY

Your customized settings for QGIS are stored based on the operating system.  , the settings are stored in your home directory in `.qt/qgisrc`.  , the settings are stored in the registry. Depending on your computing environment, storing passwords in your QGIS settings may be a security risk.

5.2.2 Loading a PostGIS Layer

 Once you have one or more connections defined, you can load layers from the PostgreSQL database. Of course this requires having data in PostgreSQL. See Section 5.2.4 for a discussion on importing data into the database.

To load a layer from PostGIS, perform the following steps:

- If the **Add PostGIS Table(s)** dialog is not already open, click on the  **Add a PostGIS Layer** toolbar button.

- Choose the connection from the drop-down list and click **Connect**.
- Find the layer you wish to add in the list of available layers.
- Select it by clicking on it. You can select multiple layers by holding down the **shift** key while clicking. See Section 5.5 for information on using the PostgreSQL Query Builder to further define the layer.
- Click on the **Add** button to add the layer to the map.

Tip 9 POSTGIS LAYERS

Normally a PostGIS layer is defined by an entry in the `geometry_columns` table. From version 0.11.0 on, QGIS can load layers that do not have an entry in the `geometry_columns` table. This includes both tables and views. Defining a spatial view provides a powerful means to visualize your data. Refer to your PostgreSQL manual for information on creating views.

5.2.3 Some details about PostgreSQL layers

This section contains some details on how QGIS accesses PostgreSQL layers. Most of the time QGIS should simply provide you with a list of database tables that can be loaded, and load them on request. However, if you have trouble loading a PostgreSQL table into QGIS, the information below may help you understand any QGIS messages and give you direction on changing the PostgreSQL table or view definition to allow QGIS to load it.

QGIS requires that PostgreSQL layers contain a column that can be used as a unique key for the layer. For tables this usually means that the table needs a primary key, or a column with a unique constraint on it. QGIS additionally requires that this column be of type `int4` (an integer of size 4 bytes). If a table lacks these items, the `oid` column will be used instead. Performance will be improved if the column is indexed (note that primary keys are automatically indexed in PostgreSQL).

If the PostgreSQL layer is a view, the same requirements exist, but views don't have primary keys or columns with unique constraints on them. In this case QGIS will try to find a column in the view that is derived from a table column that is suitable. If one cannot be found, QGIS will not load the layer. If this occurs, the solution is to alter the view so that it does include a suitable column (a type of `int4` and either a primary key or with a unique constraint, preferably indexed).

5.2.4 Importing Data into PostgreSQL

shp2pgsql

Data can be imported into PostgreSQL using a number of methods. PostGIS includes a utility called `shp2pgsql` that can be used to import shapefiles into a PostGIS enabled database. For example, to

import a shapefile named `lakes.shp` into a PostgreSQL database named `gis_data`, use the following command:


```
shp2pgsql -s 2964 lakes.shp lakes_new | psql gis_data
```


This creates a new layer named `lakes_new` in the `gis_data` database. The new layer will have a spatial reference identifier (SRID) of 2964. See Section 8 for more information on spatial reference systems and projections.

Tip 10 EXPORTING DATASETS FROM POSTGIS

Like the import-tool `shp2pgsql` there is also a tool to export PostGIS-datasets as shapefiles: `pgsql2shp`. This is shipped within your PostGIS distribution.

SPIT Plugin

 QGIS comes with a plugin named SPIT (Shapefile to PostGIS Import Tool). SPIT can be used to load multiple shapefiles at one time and includes support for schemas. To use SPIT, open the Plugin Manager from the **Plugins** menu, check the box next to the **SPIT plugin** and click **OK**. The SPIT icon will be added to the plugin toolbar.

To import a shapefile, click on the  **SPIT** tool in the toolbar to open the **SPIT - Shapefile to PostGIS Import Tool** dialog. Select the PostGIS database you want to connect to and click on **Connect**. Now you can add one or more files to the queue by clicking on the **Add** button. To process the files, click on the **OK** button. The progress of the import as well as any errors/warnings will be displayed as each shapefile is processed.

Tip 11 IMPORTING SHAPEFILES CONTAINING POSTGRES SQL RESERVED WORDS

If a shapefile is added to the queue containing fields that are reserved words in the PostgreSQL database a dialog will popup showing the status of each field. You can edit the field names prior to import and change any that are reserved words (or change any other field names as desired). Attempting to import a shapefile with reserved words as field names will likely fail.

ogr2ogr


Beside `shp2pgsql` and SPIT there is another tool for feeding geodata in PostGIS: `ogr2ogr`. This is part of your GDAL installation. To import a shapefile into PostGIS, do the following:

```
ogr2ogr -f "PostgreSQL" PG:"dbname=postgis host=myhost.de user=postgres \
password=topsecret" alaska.shp
```

This will import the shapefile `alaska.shp` into the PostGIS-database `postgis` using the user `postgres` with the password `topsecret` on host `myhost.de`.

Note that OGR must be built with PostgreSQL to support PostGIS. You can see this by typing

```
ogrinfo --formats | grep -i post
```

If you like to use PostgreSQL's `COPY`-command instead of the default `INSERT INTO` method you can export the following environment-variable (at least available on  and **X**):

```
export PG_USE_COPY=YES
```

`ogr2ogr` does not create spatial indexes like `shp2pgsql` does. You need to create them manually using the normal SQL-command `CREATE INDEX` afterwards as an extra step (as described in the next section [5.2.5](#)).

5.2.5 Improving Performance

Retrieving features from a PostgreSQL database can be time consuming, especially over a network. You can improve the drawing performance of PostgreSQL layers by ensuring that a spatial index exists on each layer in the database. PostGIS supports creation of a GiST (Generalized Search Tree) index to speed up spatial searches of the data.

The syntax for creating a GiST³ index is:

```
CREATE INDEX [indexname] ON [tablename]
  USING GIST ( [geometryfield] GIST_GEOMETRY_OPS );
```

Note that for large tables, creating the index can take a long time. Once the index is created, you should perform a `VACUUM ANALYZE`. See the PostGIS documentation ([4](#)) for more information.

The following is an example of creating a GiST index:

```
gsherman@madison:~/current$ psql gis_data
Welcome to psql 8.3.0, the PostgreSQL interactive terminal.
```

```
Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help with psql commands
```

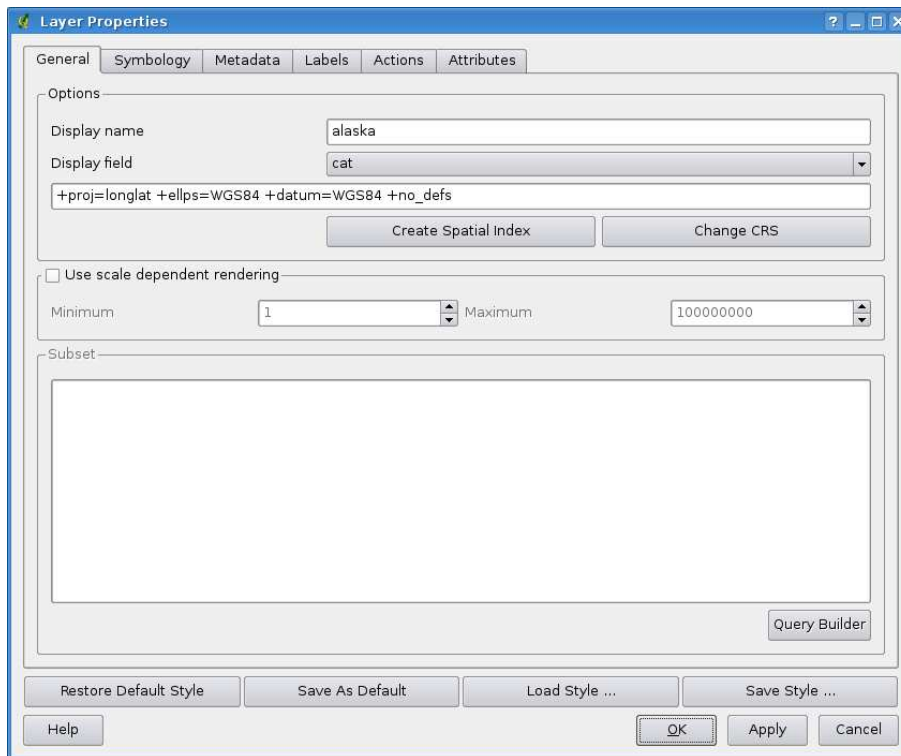
³GiST index information is taken from the PostGIS documentation available at <http://postgis.refractive.net>

```
\g or terminate with semicolon to execute query
\q to quit
```

```
gis_data=# CREATE INDEX sidx_alaska_lakes ON alaska_lakes
gis_data=# USING GIST (the_geom GIST_GEOMETRY_OPS);
CREATE INDEX
gis_data=# VACUUM ANALYZE alaska_lakes;
VACUUM
gis_data=# \q
gsherman@madison:~/current$
```

5.3 The Vector Properties Dialog

The **Layer Properties** dialog for a vector layer provides information about the layer, symbology settings and labeling options. If your vector layer has been loaded from a PostgreSQL / PostGIS datastore, you can also alter the underlying SQL for the layer - either by hand editing the SQL on the **General** tab or by invoking the **Query Builder** dialog on the **General** tab. To access the **Layer Properties** dialog, double-click on a layer in the legend or right-click on the layer and select **Properties** from the popup menu.

Figure 6: Vector Layer Properties Dialog 

5.3.1 General Tab

The **General** tab is essentially like that of the raster dialog. It allows you to change the display name, set scale dependent rendering options, create a spatial index of the vector file (only for OGR supported formats and PostGIS) and view or change the projection of the specific vector layer.

The **Query Builder** button allows you to create a subset of the features in the layer - but this button currently only is available when you open the attribute table and select the **Advanced ...** button.

5.3.2 Symbology Tab

QGIS supports a number of symbology renderers to control how vector features are displayed. Currently the following renderers are available:

Single symbol - a single style is applied to every object in the layer.

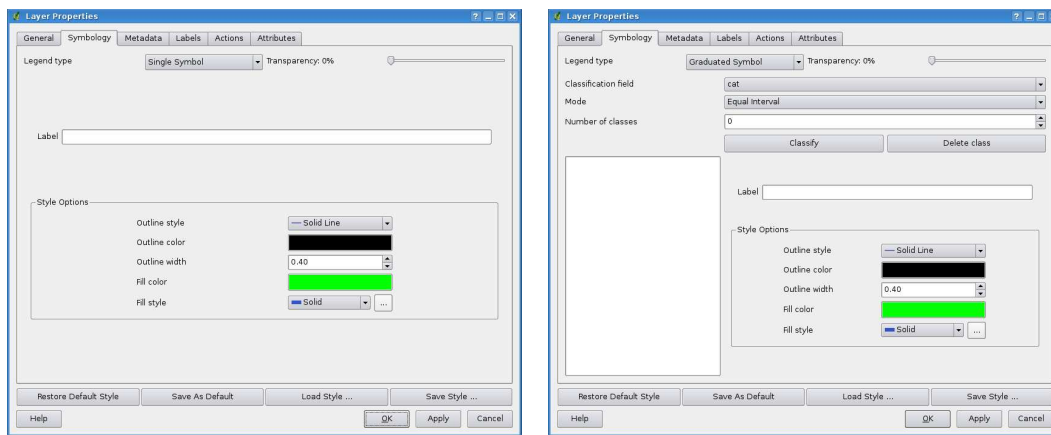
Graduated symbol - objects within the layer are displayed with different symbols classified by the values of a particular field.

Continuous color - objects within the layer are displayed with a spread of colours classified by the numerical values within a specified field.

Unique value - objects are classified by the unique values within a specified field with each value having a different symbol.

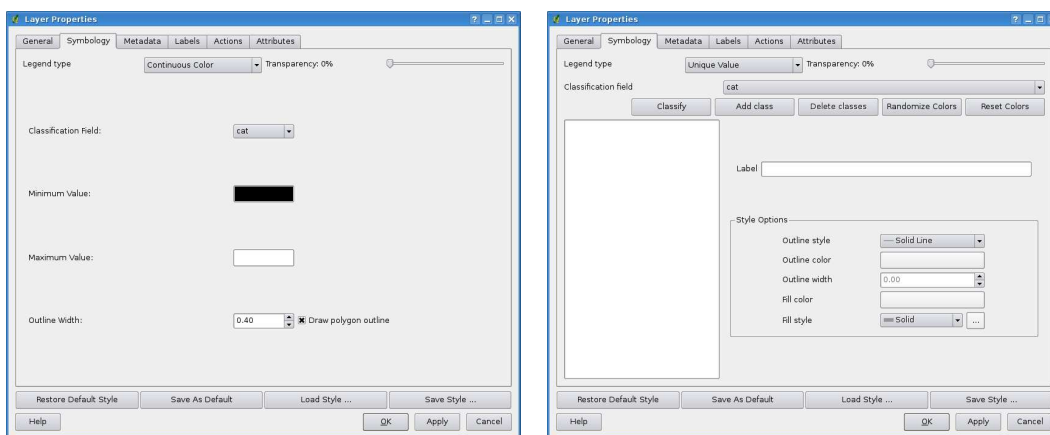
To change the symbology for a layer, simply double click on its legend entry and the vector **Layer Properties** dialog will be shown.

Figure 7: Symbolizing-options 



(a) Single symbol

(b) Graduated symbol



(c) Continuous color

(d) Unique value

Style Options

Within this dialog you can style your vector layer. Depending on the selected rendering option you have the possibility to also classify your mapfeatures.

At least the following styling options apply for nearly all renderers:

Outline style - pen-style for your outline of your feature. you can also set this to 'no pen'.

Outline color - color of the outline of your feature

Outline width - width of your features

Fill color - fill-color of your features.

Fill style - Style for filling. Beside the given brushes you can select **Fill style ? texture ▼** and click the **...** button for selecting your own fill-style. Currently the fileformats *.jpeg, *.xpm, and *.png are supported.

Once you have styled your layer you also could save your layer-style to a separate file (with *.qml-ending). To do this, use the button **Save Style ...**. No need to say that **Load Style ...** loads your saved layer-style-file.

If you wish to always use a particular style whenever the layer is loaded, use the **Save As Default** button to make your style the default. Also, if you make changes to the style that you are not happy with, use the **Restore Default Style** button to revert to your default style.

Vector transparency

QGIS 1.0.0 allows to set a transparency for every vector layer. This can be done with the slider **Transparency 0% ▼** inside the **symbology** tab (see fig. 6). This is very useful for overlaying several vector layers.

5.3.3 Metadata Tab

The **Metadata** tab contains information about the layer, including specifics about the type and location, number of features, feature type, and the editing capabilities. The Layer Spatial Reference System section, providing projection information, and the Attribute field info section, listing fields and their data types, are displayed on this tab. This is a quick way to get information about the layer.

5.3.4 Labels Tab

The **Labels** tab allows you to enable labeling features and control a number of options related to fonts, placement, style, alignment and buffering.

We will illustrate this by labelling the lakes shapefile of the `qgis_example_dataset`:

1. Load the Shapefile `alaska.shp` and GML file `lakes.gml` in QGIS.
2. Zoom in a bit to your favorite area with some lake.
3. Make the `lakes` layer active.

4. Open the **Layer Properties** dialog.
5. Click on the **Labels** tab.
6. Check the **Display labels** checkbox to enable labeling.
7. Choose the field to label with. We'll use **Field containing label NAMES**.
8. Enter a default for lakes that have no name. The default label will be used each time QGIS encounters a lake with no value in the NAMES field.
9. Click **Apply**.

Now we have labels. How do they look? They are probably too big and poorly placed in relation to the marker symbol for the lakes.

Select the **Font** entry and use the **Font** and **Color** buttons to set the font and color. You can also change the angle and the placement of the text-label.

To change the position of the text relative to the feature:

1. Click on the **Font** entry.
2. Change the placement by selecting one of the radio buttons in the **Placement** group. To fix our labels, choose the **Right** radio button.
3. the **Font size units** allows you to select between **Points** or **Map units**.
4. Click **Apply** to see your changes without closing the dialog.

Things are looking better, but the labels are still too close to the marker. To fix this we can use the options on the **Position** entry. Here we can add offsets for the X and Y directions. Adding an X offset of 5 will move our labels off the marker and make them more readable. Of course if your marker symbol or font is larger, more of an offset will be required.

The last adjustment we'll make is to **buffer** the labels. This just means putting a backdrop around them to make them stand out better. To buffer the lakes labels:

1. Click the **Buffer** tab.
2. Click the **Buffer Labels?** checkbox to enable buffering.
3. Choose a size for the buffer using the spin box.
4. Choose a color by clicking on **Color** and choosing your favorite from the color selector. You can also set some transparency for the buffer if you prefer.
5. Click **Apply** to see if you like the changes.

If you aren't happy with the results, tweak the settings and then test again by clicking **Apply**.

A buffer of 1 points seems to give a good result. Notice you can also specify the buffer size in map units if that works out better for you.

The remaining entries inside the **Label** tab allow you control the appearance of the labels using attributes stored in the layer. The entries beginning with **Data defined** allow you to set all the parameters for the labels using fields in the layer.

Not that the **Label** tab provides a **preview-box** where your selected label is shown.

5.3.5 Actions Tab

QGIS provides the ability to perform an action based on the attributes of a feature. This can be used to perform any number of actions, for example, running a program with arguments built from the attributes of a feature or passing parameters to a web reporting tool.

Actions are useful when you frequently want to run an external application or view a web page based on one or more values in your vector layer. An example is performing a search based on an attribute value. This concept is used in the following discussion.

Defining Actions

Attribute actions are defined from the vector **Layer Properties** dialog. To define an action, open the vector **Layer Properties** dialog and click on the **Actions** tab. Provide a descriptive name for the action. The action itself must contain the name of the application that will be executed when the action is invoked. You can add one or more attribute field values as arguments to the application. When the action is invoked any set of characters that start with a % followed by the name of a field will be replaced by the value of that field. The special characters %% will be replaced by the value of the field that was selected from the identify results or attribute table (see Using Actions below). Double quote marks can be used to group text into a single argument to the program, script or command. Double quotes will be ignored if preceded by a backslash.

If you have field names that are substrings of other field names (e.g., `col1` and `col10`) you should indicate so, by surrounding the field name (and the % character) with square brackets (e.g., `[%col10]`). This will prevent the `%col10` field name being mistaken for the `%col1` field name with a 0 on the end. The brackets will be removed by QGIS when it substitutes in the value of the field. If you want the substituted field to be surrounded by square brackets, use a second set like this: `[[%col10]]`.

The **Identify Results** dialog box includes a *(Derived)* item that contains information relevant to the layer type. The values in this item can be accessed in a similar way to the other fields by using preceding the derived field name by *(Derived)*.. For example, a point layer has an X and Y field and the value of these can be used in the action with `%(Derived).X` and `%(Derived).Y`. The derived attributes are only available from the **Identify Results** dialog box, not the **Attribute Table** dialog box.

Two example actions are shown below:



- konqueror http://www.google.com/search?q=%nam
- konqueror http://www.google.com/search?q=%%

In the first example, the web browser konqueror is invoked and passed a URL to open. The URL performs a Google search on the value of the `nam` field from our vector layer. Note that the application or script called by the action must be in the path or you must provide the full path. To be sure, we could rewrite the first example as: `/opt/kde3/bin/konqueror http://www.google.com/search?q=%nam`. This will ensure that the konqueror application will be executed when the action is invoked.

The second example uses the `%%` notation which does not rely on a particular field for its value. When the action is invoked, the `%%` will be replaced by the value of the selected field in the identify results or attribute table.


Using Actions

Actions can be invoked from either the **Identify Results** dialog or an **Attribute Table** dialog.

(Recall that these dialogs can be opened by clicking  **Identify Features** or  **Open Table**.)

To invoke an action, right click on the record and choose the action from the popup menu. Actions are listed in the popup menu by the name you assigned when defining the actions. Click on the action you wish to invoke.

If you are invoking an action that uses the `%%` notation, right-click on the field value in the **Identify Results** dialog or the **Attribute Table** dialog that you wish to pass to the application or script.

Here is another example that pulls data out of a vector layer and inserts them into a file using bash and the `echo` command (so it will only work  or perhaps **X**). The layer in question has fields for a species name `taxon_name`, latitude `lat` and longitude `long`. I would like to be able to make a spatial selection of a localities and export these field values to a text file for the selected record (shown in yellow in the QGIS map area). Here is the action to achieve this:

```
bash -c "echo \"%taxon_name %lat %long\" >> /tmp/species_localities.txt"
```

After selecting a few localities and running the action on each one, opening the output file will show something like this:

```
Acacia mearnsii -34.0800000000 150.0800000000
Acacia mearnsii -34.9000000000 150.1200000000
Acacia mearnsii -35.2200000000 149.9300000000
Acacia mearnsii -32.2700000000 150.4100000000
```

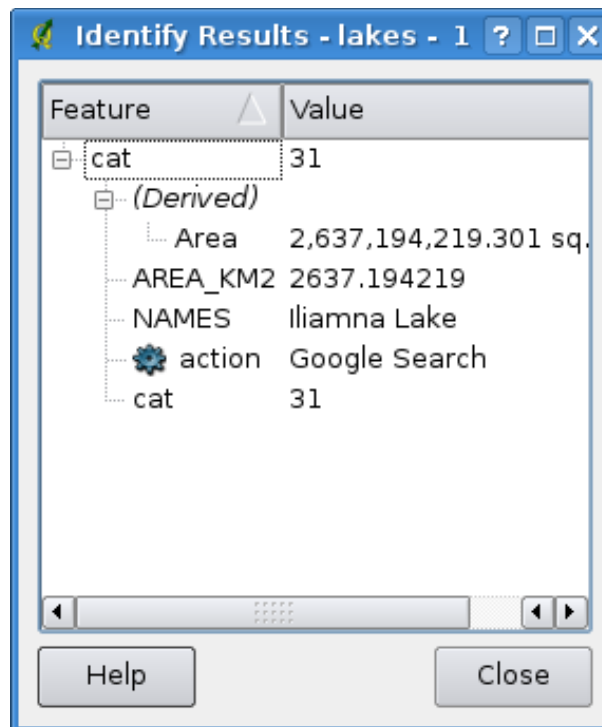
As an exercise we create an action that does a Google search on the `lakes` layer. First we need to determine the URL needed to perform a search on a keyword. This is easily done by just going to Google and doing a simple search, then grabbing the URL from the address bar in your browser. From this little effort we see that the format is: `http://google.com/search?q=qgis`, where `qgis` is the search term. Armed with this information, we can proceed:

1. Make sure the `lakes` layer is loaded.
2. Open the **Layer Properties** dialog by double-clicking on the layer in the legend or right-click and choose **Properties** from the popup menu.
3. Click on the **Actions** tab.
4. Enter a name for the action, for example `Google Search`.
5. For the action, we need to provide the name of the external program to run. In this case, we can use `Firefox`. If the program is not in your path, you need to provide the full path.
6. Following the name of the external application, add the URL used for doing a Google search, up to but not included the search term: `http://google.com/search?q=`
7. The text in the Action field should now look like this:
`firefox http://google.com/search?q=`
8. Click on the drop-down box containing the field names for the `lakes` layer. It's located just to the left of the **Insert Field** button.
9. From the drop-down box, select **NAMES** and click **Insert Field**.
10. Your action text now looks like this:
`firefox http://google.com/search?q=%NAMES`
11. To finalize the action click the **Insert action** button.

This completes the action and it is ready to use. The final text of the action should look like this:

```
firefox http://google.com/search?q=%NAMES
```

We can now use the action. Close the **Layer Properties** dialog and zoom in to an area of interest. Make sure the `lakes` layer is active and identify a lake. In the result box you'll now see that our action is visible:

Figure 8: Select feature and choose action 

When we click on the action, it brings up Firefox and navigates to the URL <http://www.google.com/search?q=Tustumena>. It is also possible to add further attribute fields to the action. Therefore you can add a “+” to the end of the action text, select another field and click on **Insert Field**. In this example there is just no other field available that would make sense to search for.

You can define multiple actions for a layer and each will show up in the **Identify Results** dialog. You can also invoke actions from the attribute table by selecting a row and right-clicking, then choosing the action from the popup menu.

You can think of all kinds of uses for actions. For example, if you have a point layer containing locations of images or photos along with a file name, you could create an action to launch a viewer to display the image. You could also use actions to launch web-based reports for an attribute field or combination of fields, specifying them in the same way we did in our Google search example.

5.3.6 Attributes Tab

Within the **Attributes** tab the attributes of the selected dataset can be manipulated. The buttons **New Column** and **Delete Column** can be used, when the dataset is in editing mode. At the

moment only columns from PostGIS layers can be edited, because this feature is not yet supported by the OGR library.

The **Toggle editing mode** button toggles this mode.

edit widget

Within the **Attributes** tab you also find an `edit widget` and a `value` column. These two columns can be used to define values or a range of values that are allowed to be added to the specific attribute table columns. They are used to produce different edit widgets in the attribute dialog. These widgets are:

- line edit: an edit field which allows to enter simple text (or restrict to numbers for numeric attributes).
- unique value: a list of unique attribute values of all pre-existing features is produced and presented in a combo box for selection.
- unique value (editable): a combination of 'line edit' and 'unique value'. The edit field completes entered values to the unique value, but also allows to enter new values.
- value map: a combobox to select from a set of values specified in the `value` column the **Attributes** tab. The possible values are delimited by a semicolon (e.g. `high;medium;low`). It is also possible to prepend a label to each value, which is delimited with an equal sign (e.g. `high=1;medium=2;low=3`). The label is shown in the combobox instead of the value.
- classification: if a unique value renderer is selected for the layer, the values used for the classes are presented for selection in a combobox.
- range (editable): A edit field that allows to restrict numeric values to a given range. That range is specified by entering minimum and maximum value delimited by a semicolon (e.g. `0;360`) in the `value` column of the **Attributes** tab.
- range (slider): A slider widget is presented that allows selection of a value in a given range and precision. The range is specified by minimum, maximum value and a step width (e.g. `0;360;10`) in the `value` column of the **Attributes** tab.
- file name: the line edit widget is accompanied by a push button. When pressed it allows to select a filename using the standard file dialog.

5.4 Editing

QGIS supports basic capabilities for editing vector geometries. Before reading any further you should note that at this stage editing support is still preliminary. Before performing any edits, always make a backup of the dataset you are about to edit.

Note - the procedure for editing GRASS layers is different - see Section [9.7](#) for details.

5.4.1 Setting the Snapping Tolerance and Search Radius

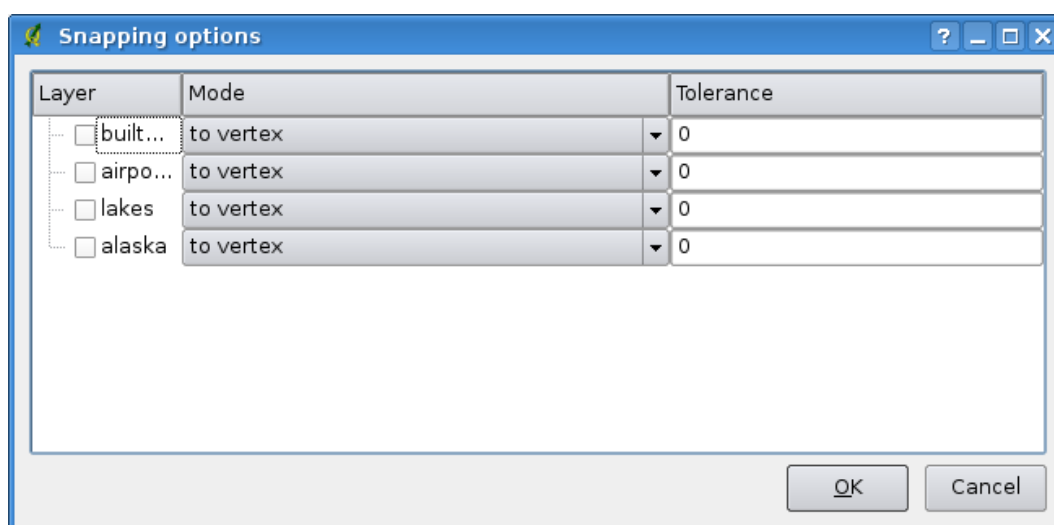
Before we can edit vertices, it is very important to set the snapping tolerance and search radius to a value that allows us an optimal editing of the vector layer geometries.

Snapping tolerance

Snapping tolerance is the distance QGIS uses to search for the closest vertex and/or segment you are trying to connect when you set a new vertex or move an existing vertex. If you aren't within the snap tolerance, QGIS will leave the vertex where you release the mouse button, instead of snapping it to an existing vertex and/or segment.

1. A general, project wide snapping tolerance can be defined choosing **Settings** > **Options**. In the **Digitizing** tab you can select between to vertex, to segment or to vertex and segment as default snap mode. You can also define a default snapping tolerance and a search radius for vertex edits. Remember the tolerance is in layer units. In our digitizing project (working with the Alaska dataset), the units are in feet. Your results may vary, but something on the order of 300ft should be fine at a scale of 1:10 000 should be a reasonable setting.
2. A layer based snapping tolerance can be defined by choosing **Settings** > **Project Properties...**. In the **General** tab, section **Digitize** you can click on **Snapping options...** to enable and adjust snapping mode and tolerance on a layer basis (see Figure 9).

Figure 9: Edit snapping options on a layer basis 



Search radius

Search radius is the distance QGIS uses to search for the closest vertex you are trying to move when you click on the map. If you aren't within the search radius, QGIS won't find and select any vertex for editing and it will pop up an annoying warning to that effect. Snap tolerance and search radius are set in map units so you may find you need to experiment to get them set right. If you specify too big of a tolerance, QGIS may snap to the wrong vertex, especially if you are dealing with a large number of vertices in close proximity. Set search radius too small and it won't find anything to move.

The search radius for vertex edits in layer units can be defined in the **Digitizing** tab under **Settings** > **Options**. The same place where you define the general, project wide snapping tolerance.

5.4.2 Topological editing

Besides layer based snapping options the **General** tab in menu **Settings** -> **Project Properties...** also provides some topological functionalities. In the Digitizing option group you can **Enable topological editing** and/or activate **Avoid intersections of new polygons**.

Enable topological editing

The option **Enable topological editing** is for editing and maintaining common boundaries in polygon mosaics. QGIS "detects" a shared boundary in a polygon mosaic and you only have to move the vertex once and QGIS will take care about updating the other boundary.

Avoid intersections of new polygons

The second topological option called **Avoid intersections of new polygons** avoids overlaps in polygon mosaics. It is for quicker digitizing of adjacent polygons. If you already have one polygon, it is possible with this option to digitise the second one such that both intersect and qgis then cuts the second polygon to the common boundary. The advantage is that users don't have to digitize all vertices of the common boundary.

5.4.3 Editing an Existing Layer

By default, QGIS loads layers read-only: This is a safeguard to avoid accidentally editing a layer if there is a slip of the mouse. However, you can choose to edit any layer as long as the data provider supports it, and the underlying data source is writable (i.e. its files are not read-only).

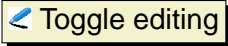

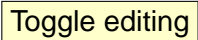
Layer editing is most versatile when used on PostgreSQL/PostGIS data sources.

Tip 12 DATA INTEGRITY

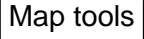

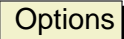
It is always a good idea to back up your data source before you start editing. While the authors of QGIS have made every effort to preserve the integrity of your data, we offer no warranty in this regard.

Tip 13 MANIPULATING ATTRIBUTE DATA

Currently only PostGIS layers are supported for adding or dropping attribute columns within this dialog. In future versions of QGIS, other datasources will be supported, because this feature was recently implemented in GDAL/OGR > 1.6.0

All editing sessions start by choosing the  option. This can be found in the context menu after right clicking on the legend entry for that layer. Alternately, you can use the   button from the toolbar to start or stop the editing mode. Once the layer is in edit mode, markers will appear at the vertices, and additional tool buttons on the editing toolbar will become available.

Zooming with the mouse wheel






While digitizing you can use the mouse wheel to zoom in and out on the map. Place the mouse cursor inside the map area and roll it forward (away from you) to zoom in and backwards (towards you) to zoom out. The mouse cursor position will be the center of the zoomed area of interest. You can customize the behavior of the mouse wheel zoom using the  tab under the  >  menu.

Panning with the arrow keys


Panning the Map during digitizing is possible with the arrow keys. Place the mouse cursor inside the map area and click on the right arrow key to pan east, left arrow key to pan west, up arrow key to pan north and down arrow key to pan south.

You can also use the spacebar to temporarily cause mouse movements to pan then map. The PgUp and PgDown keys on your keyboard will cause the map display to zoom in or out without interrupting your digitising session.

You can perform the following editing functions:










- Add Features:  Capture Point,  Capture Line and  Capture Polygon
-  Add Ring
-  Add Island

Tip 14 SAVE REGULARLY




Remember to toggle  **Toggle editing** off regularly. This allows you to save your recent changes, and also confirms that your data source can accept all your changes.




Tip 15 CONCURRENT EDITS

This version of QGIS does not track if somebody else is editing a feature at the same time as you. The last person to save their edits wins.

-  Split Features
-  Move Features
-  Move Vertex
-  Add Vertex
-  Delete Vertex
-  Delete Selected
-  Cut Features
-  Copy Features
-  Paste Features

Adding Features

Before you start adding features, use the  **pan** and  **zoom-in** /  **zoom-out** tools to first navigate to the area of interest.

Then you can use the  **Capture point**,  **Capture line** or  **Capture polygon** icons on the toolbar to put the QGIS cursor into digitizing mode.




For each feature, you first digitize the geometry, then enter its attributes.

To digitize the geometry, left-click on the map area to create the first point of your new feature.

Tip 16 ZOOM IN BEFORE EDITING

Before editing a layer, you should zoom in to your area of interest. This avoids waiting while all the vertex markers are rendered across the entire layer.

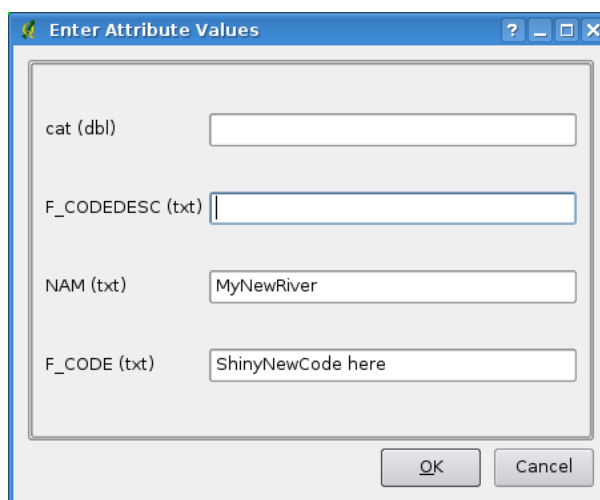
Tip 17 VERTEX MARKERS

The current version of QGIS supports two kinds of vertex-markers - a semi-transparent circle or a cross. To change the marker style, choose  Options from the  Settings menu and click on the  Digitizing tab and select the appropriate entry.

For lines and polygons, keep on left-clicking for each additional point you wish to capture. When you have finished adding points, right-click anywhere on the map area to confirm you have finished entering the geometry of that feature.

The attribute window will appear, allowing you to enter the information for the new feature. Figure 10 shows setting attributes for a fictitious new river in Alaska.

Figure 10: Enter Attribute Values Dialog after digitizing a new vector feature 

**Move Feature**

You can move features using the  Move Feature icon on the toolbar.

Split Feature

You can split features using the  Split Features icon on the toolbar.

Tip 18 ATTRIBUTE VALUE TYPES

At least for shapefile editing the attribute types are validated during the entry. Because of this, it is not possible to enter a number into the text-column in the dialog **Enter Attribute Values** or vica versa. If you need to do so, you should edit the attributes in a second step within the **Attribute table** dialog.

Editing Vertices of a Feature


For both PostgreSQL/PostGIS and shapefile-based layers, the vertices of features can be edited.

Vertices can be directly edited, that is, you don't have to choose which feature to edit before you can change its geometry. In some cases, several features may share the same vertex and so the following rules apply when the mouse is pressed down near map features:

- **Lines** - The nearest line to the mouse position is used as the target feature. Then (for moving and deleting a vertex) the nearest vertex on that line is the editing target.
- **Polygons** - If the mouse is inside a polygon, then it is the target feature; otherwise the nearest polygon is used. Then (for moving and deleting a vertex) the nearest vertex on that polygon is the editing target.

You will need to set the property **Settings** > **Options** > **Digitizing** > **Search Radius** to a number greater than zero. Otherwise QGIS will not be able to tell which feature is being edited.

Adding Vertices of a Feature

You can add new vertices to a feature by using the  **Add Vertex** icon on the toolbar.

Note, it doesn't make sense to add more vertices to a Point feature!

In this version of QGIS, vertices can only be added to an *existing* line segment of a line feature. If you want to extend a line beyond its end, you will need to move the terminating vertex first, then add a new vertex where the terminus used to be.

Moving Vertices of a Feature

You can move vertices using the  **Move Vertex** icon on the toolbar.

Deleting Vertices of a Feature


You can delete vertices by using the  **Delete Vertex** icon on the toolbar.

Note, it doesn't make sense to delete the vertex of a Point feature! Delete the whole feature instead.


Similarly, a one-vertex line or a two-vertex polygon is also fairly useless and will lead to unpredictable results elsewhere in QGIS, so don't do that.

Warning: A vertex is identified for deletion as soon as you click the mouse near an eligible feature. To undo, you will need to toggle Editing off and then discard your changes. (Of course this will mean that other unsaved changes will be lost, too.)


Add Ring

You can create ring polygons using the  **Add Ring** icon in the toolbar. This means inside an existing area it is possible to digitize further polygons, that will occur as a 'whole', so only the area in between the boundaries of the outer and inner polygons remain as a ring polygon.

Add Island

You can  **add island** polygons to a selected multipolygon. The new island polygon has to be digitized outside the selected multipolygon.


Cutting, Copying and Pasting Features



Selected features can be cut, copied and pasted between layers in the same QGIS project, as long as destination layers are set to  **Toggle editing** beforehand.

Features can also be pasted to external applications as text: That is, the features are represented in CSV format with the geometry data appearing in the OGC Well-Known Text (WKT) format.

However in this version of QGIS, text features from outside QGIS cannot be pasted to a layer within QGIS. When would the copy and paste function come in handy? Well, it turns out that you can edit more than one layer at a time and copy/paste features between layers. Why would we want to do this? Say we need to do some work on a new layer but only need one or two lakes, not the 5,000 on our `big_lakes` layer. We can create a new layer and use copy/paste to plop the needed lakes into it.

As an example we are copying some lakes to a new layer:

1. Load the layer you want to copy from (source layer)
2. Load or create the layer you want to copy to (target layer)
3. Start editing for both layers
4. Make the source layer active by clicking on it in the legend
5. Use the  **Select** tool to select the feature(s) on the source layer

6. Click on the  **Copy Features** tool
7. Make the destination layer active by clicking on it in the legend
8. Click on the  **Paste Features** tool
9. Stop editing and save the changes

What happens if the source and target layers have different schemas (field names and types are not the same)? QGIS populates what matches and ignores the rest. If you don't care about the attributes being copied to the target layer, it doesn't matter how you design the fields and data types. If you want to make sure everything - feature and its attributes - gets copied, make sure the schemas match.

Tip 19 CONGRUENCY OF PASTED FEATURES

If your source and destination layers use the same projection, then the pasted features will have geometry identical to the source layer. However if the destination layer is a different projection then QGIS cannot guarantee the geometry is identical. This is simply because there are small rounding-off errors involved when converting between projections.


Deleting Selected Features

If we want to delete an entire polygon, we can do that by first selecting the polygon using the regular




Select Features

tool. You can select multiple features for deletion. Once you have the selec-

tion set, use the  **Delete Selected** tool to delete the features. There is no undo function, but remember your layer isn't really changed until you stop editing and choose to save your changes. So if you make a mistake, you can always cancel the save.



Cut Features

tool on the digitizing toolbar can also be used to delete features. This effectively deletes the feature but also places it on a "spatial clipboard". So we cut the feature to delete. We could then use the  **paste tool** to put it back, giving us a one-level undo capability. Cut, copy, and paste work on the currently selected features, meaning we can operate on more than one at a time.

Tip 20 FEATURE DELETION SUPPORT

When editing ESRI shapefiles, the deletion of features only works if QGIS is linked to a GDAL version 1.3.2 or greater. The OS X and Windows versions of QGIS available from the download site are built using GDAL 1.3.2 or higher.

Snap Mode


QGIS allows digitized vertices to be snapped to other vertices of the same layer. To set the snapping tolerance, go to **Settings** > **Options** -> **Digitizing**. Note that the snapping tolerance is in map units.

Saving Edited Layers

When a layer is in editing mode, any changes remain in the memory of QGIS. Therefore they are not committed/saved immediately to the data source or disk. When you turn editing mode off (or quit QGIS for that matter), you are then asked if you want to save your changes or discard them.

If the changes cannot be saved (e.g. disk full, or the attributes have values that are out of range), the QGIS in-memory state is preserved. This allows you to adjust your edits and try again.

5.4.4 Creating a New Layer

To create a new layer for editing, choose  **New Vector Layer** from the **Layer** menu. The **New Vector Layer** dialog will be displayed as shown in Figure 11. Choose the type of layer (point, line or polygon).

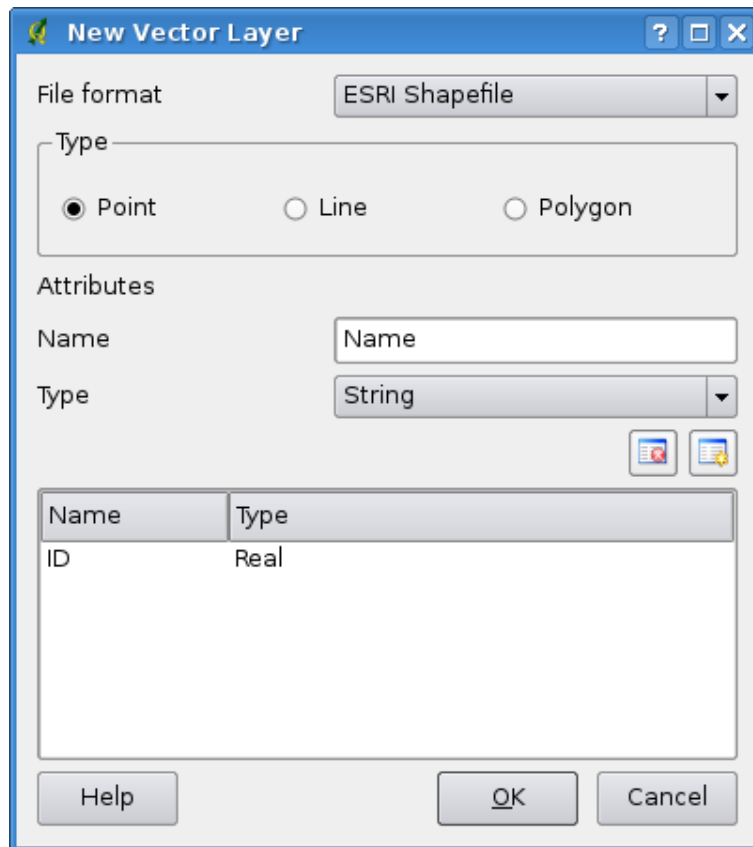

Note that QGIS does not yet support creation of 2.5D features (i.e. features with X,Y,Z coordinates) or measure features. At this time, only shapefiles can be created. In a future version of QGIS, creation of any OGR or PostgreSQL layer type will be supported.

Creation of GRASS-layers is supported within the GRASS-plugin. Please refer to section 9.6 for more information on creating GRASS vector layers.

To complete the creation of the new layer, add the desired attributes by clicking on the **Add** button and specifying a name and type for the attribute. Only **Type real**, **Type integer**, and **Type string** attributes are supported. Once you are happy with the attributes, click **OK** and provide a name for the shapefile. QGIS will automatically add a `.shp` extension to the name you specify. Once the layer has been created, it will be added to the map and you can edit it in the same way as described in Section 5.4.3 above.

5.5 Query Builder

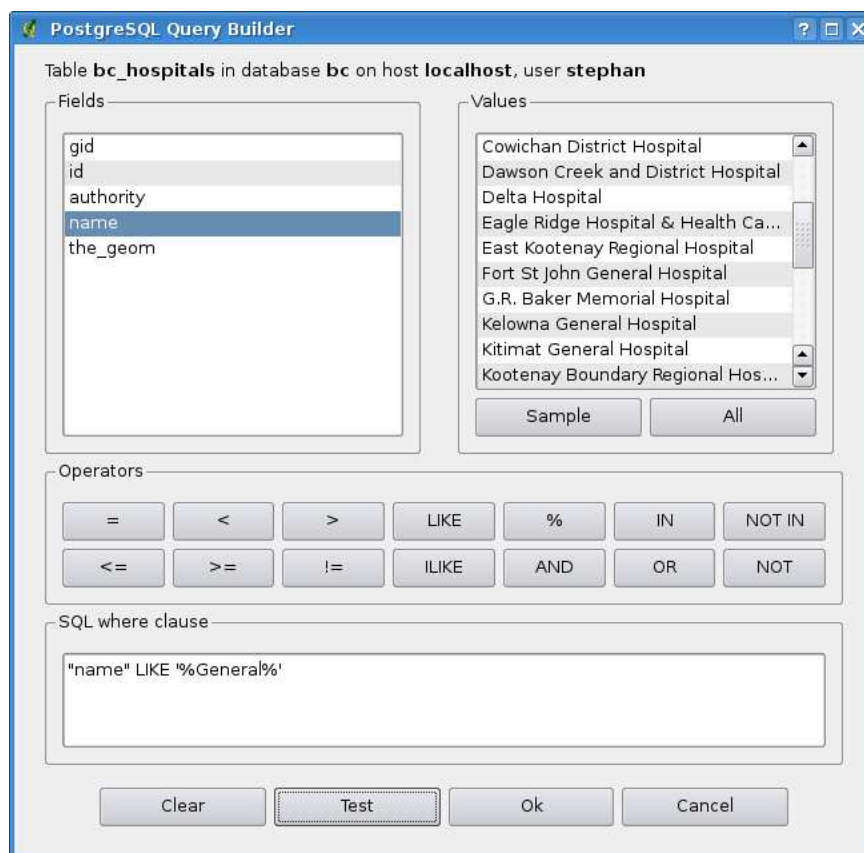
The Query Builder allows you to define a subset of a table and display it as a layer in QGIS. It can currently only be used with PostGIS layers. For example, if you have a `towns` layer with a `population` field you could select only larger towns by entering `population > 100000` in the SQL box of the query

Figure 11: Creating a New Vector Dialog 

builder. Figure 12 shows an example of the query builder populated with data from a PostGIS layer with attributes stored in PostgreSQL.

The query builder lists the layer's database fields in the list box on the left. You can get a sample of the data contained in the highlighted field by clicking on the **Sample** button. This retrieves the first 25 distinct values for the field from the database. To get a list of all possible values for a field, click on the **All** button. To add a selected field or value to the query, double-click on it. You can use the various buttons to construct the query or you can just type it into the SQL box.


To test a query, click on the **Test** button. This will return a count of the number of records that will be included in the layer. When satisfied with the query, click **OK**. The SQL for the where clause will be shown in the SQL column of the layer list.

Figure 12: Query Builder **Tip 21** CHANGING THE LAYER DEFINITION

You can change the layer definition after it is loaded by altering the SQL query used to define the layer. To do this, open the vector **Layer Properties** dialog by double-clicking on the layer in the legend and click on the **Query Builder** button on the **General** tab. See Section 5.3 for more information.

5.6 Select by query

With QGIS it is possible also to select features using a similar query builder interface to that used in 5.5. In the above section the purpose of the query builder is to only show features meeting the filter criteria as a 'virtual layer' / subset. The purpose of the select by query function is to highlight all features that meet a particular criteria. Select by query can be used with all vector data providers.

To do a 'select by query' on a loaded layer, click on the button  **Open Table** to open the attribute table of the layer. Then click the **Advanced...** button at the bottom. This starts the Query Builder that allows to define a subset of a table and display it as described in Section 5.5.

6 Working with Raster Data

This Section describes how to visualize and set raster layer properties. QGIS supports a number of different raster formats. Currently tested formats include:

- Arc/Info Binary Grid
- Arc/Info ASCII Grid
- GRASS Raster
- GeoTIFF
- JPEG
- Spatial Data Transfer Standard Grids (with some limitations)
- USGS ASCII DEM
- Erdas Imagine

Because the raster implementation in QGIS is based on the GDAL library, other raster formats implemented in GDAL are also likely to work - if in doubt try to open a sample and see if it is supported. You find more details about GDAL supported formats in Appendix [A.2](#) or at http://www.gdal.org/formats_list.html. If you want to load GRASS raster data, please refer to Section [9.2](#).


6.1 What is raster data?

Raster data in GIS are matrices of discrete cells that represent features on, above or below the earth's surface. Each cell in the raster grid is the same size, and cells are usually rectangular (in QGIS they will always be rectangular). Typical raster datasets include remote sensing data such as aerial photography or satellite imagery and modelled data such as an elevation matrix.

Unlike vector data, raster data typically do not have an associated database record for each cell. They are geocoded by its pixel resolution and the x/y coordinate of a corner pixel of the raster layer. This allows QGIS to position the data correctly in the map canvas.

QGIS makes use of georeference information inside the raster layer (e.g. GeoTiff) or in an appropriate world file to properly display the data.

6.2 Loading raster data in QGIS

Raster layers are loaded either by clicking on the  **Load Raster** icon or by selecting the **View** > **Add Raster Layer** menu option. More than one layer can be loaded at the same

time by holding down the **Control** or **Shift** key and clicking on multiple items in the dialog **Open a GDAL Supported Raster Data Source**.

Once a raster layer is loaded in the map legend you can click on the layer name with the right mouse button to select and activate layer specific features or to open a dialog to set raster properties for the layer.

Right mouse button menu for raster layers

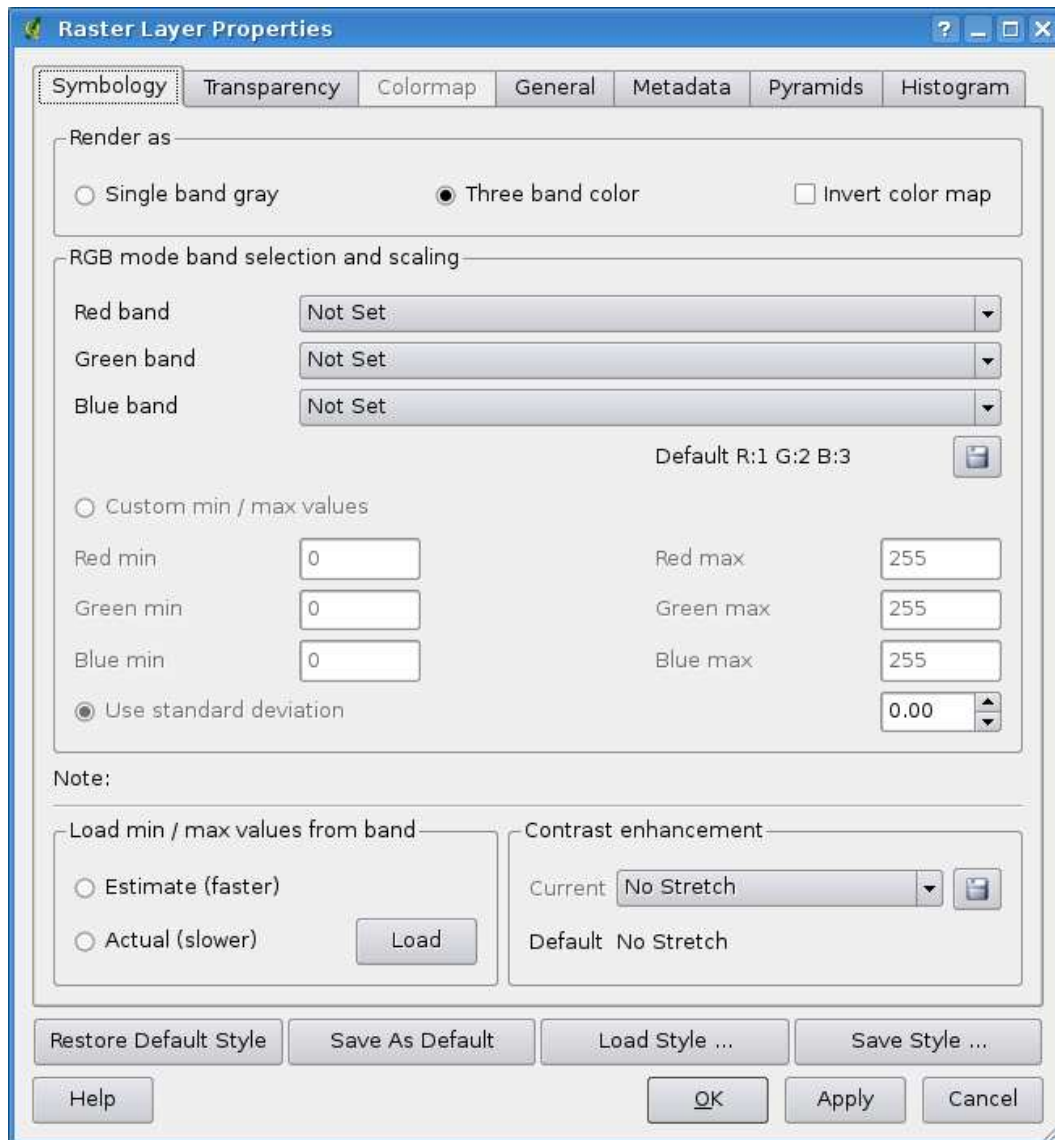
- **Zoom to layer extent**
- **Zoom to best scale (100%)**
- **Show in overview**
- **Remove**
- **Properties**
- **Rename**
- **Add Group**
- **Expand all**
- **Collapse all**
- **Show file groups**

6.3 Raster Properties Dialog

To view and set the properties for a raster layer, double click on the layer name in the map legend or right click on the layer name and choose **Properties** from the context menu: Figure 13 shows the

Raster Layer Properties dialog. There are several tabs on the dialog:

- **Symbology**
- **Transparency**
- **Colormap**
- **General**
- **Metadata**
- **Pyramids**
- **Histogram**

Figure 13: Raster Layers Properties Dialog 

6.3.1 Symbology Tab

QGIS can render raster layers in two different ways :

- Single band - one band of the image will be rendered as gray or in pseudocolors.
- Three band color - three bands from the image will be rendered, each band representing the red, green or blue component that will be used to create a color image.

Within both rendertypes you can invert the color output using the Invert color map checkbox.

Single Band Rendering

This selection offers you two possibilities to choose. At first you can select which band you like to use for rendering (if the dataset has more than one band).

The second option offers a selection of available colortables for rendering.

The following settings are available through the dropdownbox , where grayscale is the default setting. Also available are

- Pseudocolor
- Freak Out
- Colormap

When selecting the entry , the tab becomes available. See more on that at chapter 6.3.3.

QGIS can restrict the data displayed to only show cells whose values are within a given number of standard deviations of the mean for the layer. This is useful when you have one or two cells with abnormally high values in a raster grid that are having a negative impact on the rendering of the raster. This option is only available for pseudocolor images.

Three band color

This selection offers you a wide range of options to modify the appearance of your rasterlayer. For example you could switch color-bands from the standard RGB-order to something else.

Also scaling of colors are available.

Tip 22 VIEWING A SINGLE BAND OF A MULTIBAND RASTER

If you want to view a single band (for example Red) of a multiband image, you might think you would set the Green and Blue bands to "Not Set". But this is not the correct way. To display the Red band, set the image type to grayscale, then select Red as the band to use for Gray.


6.3.2 Transparency Tab

QGIS has the ability to display each raster layer at varying transparency levels. Use the transparency slider to indicate to what extent the underlying layers (if any) should be visible through the current raster layer. This is very useful, if you like to overlay more than one rasterlayer, e.g. a shaded relief-map overlaid by a classified rastermap. This will make the look of the map more three dimensional.



Additionally you can enter a rastervalue, which should be treated as *NODATA*.

An even more flexible way to customize the transparency can be done in the Custom transparency options section. The transparency of every pixel can be set in this tab.

As an example we want to set the water of our example rasterfile `landcover.tif` to a transparency of 20%. The following steps are necessary:

1. Load the rasterfile `landcover`
2. Open the **properties** dialog by double-clicking on the rasterfile-name in the legend or by right-clicking and choosing **Properties** from the popup menu.
3. select the **Transparency** tab
4. Click the  **Add values manually** button. A new row will appear in the pixel-list.
5. enter the the raster-value (we use 0 here) and adjust the transparency to 20%
6. press the **Apply** button and have a look at the map

You can repeat the steps 4 and 5 to adjust more values with custom transparency.

As you can see this is quite easy set custom transparency, but it can be quite a lot of work. Therefore you can use the button  **Export to file** to save your transparency-list to a file. The button  **Import from file** loads your transparency-settings and applies them to the current rasterlayer.


6.3.3 Colormap

The **Colormap** tab is only available, when you have selected a single-band-rendering within the tab **Symbology** (see chapt. 6.3.1).

Three ways of color interpolation are available:

- Discrete
- Linear
- Exact

The button **Add Entry** adds a color to the individual color-table. Double-Clicking on the value-column lets you inserting a specific value. Double clicking on the color-column opens the dialog **Select color** where you can select a color to apply on that value.

Alternatively you can click on the button  **Load colormap from Band**, which tries to load the table from the band (if it has any).

The block Generate new color map allows you to create newly categorized colormaps. You only need to select the number of classes you need and press the button . Currently only one Classification mode is supported.

6.3.4 General Tab

The tab displays basic information about the selected raster, including the layer source and display name in the legend (which can be modified). This tab also shows a thumbnail of the layer, its legend symbol, and the palette.

Additionally scale-dependent visibility can be set in this tab. You need to check the checkbox and set an appropriate scale where your data will be displayed in the map canvas.

Also the spatial reference system is printed here as a PROJ.4-string. This can be modified by hitting the button.

6.3.5 Metadata Tab

The tab displays a wealth of information about the raster layer, including statistics about each band in the current raster layer. Statistics are gathered on a 'need to know' basis, so it may well be that a given layers statistics have not yet been collected.

This tab is mainly for information. You cannot change any values printed inside this tab. To update the statistics you need to change to tab and press the button on the bottom right, see ch. [6.3.7](#).

6.3.6 Pyramids Tab

Large resolution raster layers can slow navigation in QGIS. By creating lower resolution copies of the data (pyramids), performance can be considerably improved as QGIS selects the most suitable resolution to use depending on the level of zoom.

You must have write access in the directory where the original data is stored to build pyramids. Several resampling methods can be used to calculate the pyramids:

- Average
- Nearest Neighbour

When checking the checkbox Build pyramids internally if possible QGIS tries to build pyramids internally.

Please note that building pyramids may alter the original data file and once created they cannot be removed. If you wish to preserve a 'non-pyramided' version of your raster, make a backup copy prior to building pyramids.

6.3.7 Histogram Tab

The Histogram tab allows you to view the distribution of the bands or colors in your raster. You must first generate the raster statistics by clicking the Refresh button. You can choose which bands to display by selecting them in the list box at the bottom left of the tab. Two different chart types are allowed:

- Bar chart
- Line graph

You can define the number of chart columns to use and decide whether you want to Allow approximation or display out of range values. Once you view the histogram, you'll notice that the band statistics have been populated on the metadata tab.

Tip 23 GATHERING RASTER STATISTICS

To gather statistics for a layer, select pseudocolor rendering and click the Apply button. Gathering statistics for a layer can be time consuming. Please be patient while QGIS examines your data!

7 Working with OGC Data

QGIS supports WMS and WFS as data sources. The support is native; WFS is implemented as a plugin.

7.1 What is OGC Data

The Open Geospatial Consortium (OGC), is an international organization with more than 300 commercial, governmental, nonprofit and research organisations worldwide. Its members develop and implement standards for geospatial content and services, GIS data processing and exchange.

Describing a basic data model for geographic features an increasing number of specifications are developed to serve specific needs for interoperable location and geospatial technology, including GIS. Further information can be found under <http://www.opengeospatial.org/>.

Important OGC specifications are:

- **WMS** - Web Map Service
- **WFS** - Web Feature Service
- **WCS** - Web Coverage Service
- **CAT** - Web Catalog Service
- **SFS** - Simple Features for SQL
- **GML** - Geography Markup Language

OGC services are increasingly being used to exchange geospatial data between different GIS implementations and data stores. QGIS can now deal with three of the above specifications, being SFS (though support of the PostgreSQL / PostGIS data provider, see Section 5.2); WFS and WMS as a client.

7.2 WMS Client

7.2.1 Overview of WMS Support

QGIS currently can act as a WMS client that understands WMS 1.1, 1.1.1 and 1.3 servers. It has particularly been tested against publicly accessible servers such as DEMIS and JPL OnEarth.

WMS servers act upon requests by the client (e.g. QGIS) for a raster map with a given extent, set of layers, symbolisation style, and transparency. The WMS server then consults its local data sources, rasterizes the map, and sends it back to the client in a raster format. For QGIS this would typically be JPEG or PNG.

WMS is generically a REST (Representational State Transfer) service rather than a fully-blown Web Service. As such, you can actually take the URLs generated by QGIS and use them in a web browser to retrieve the same images that QGIS uses internally. This can be useful for troubleshooting, as there are several brands of WMS servers in the market and they all have their own interpretation of the WMS standard.

WMS layers can be added quite simply, as long as you know the URL to access the WMS server, you have a serviceable connection to that server, and the server understands HTTP as the data transport mechanism.

7.2.2 Selecting WMS Servers


The first time you use the WMS feature, there are no servers defined. You can begin by clicking

the  **Add WMS layer** button inside the toolbar, or through the **Layer** >  **Add WMS Layer...** menu.

The dialog **Add Layer(s) from a Server** for adding layers from the WMS server pops up. Fortunately you can add some servers to play with by clicking the **Add default servers** button. This will add at least three WMS servers for you to use, including the NASA (JPL) WMS server. To define a new WMS server in the **Server Connections** section, select **New**. Then enter in the parameters to connect to your desired WMS server, as listed in table 2:

Table 2: WMS Connection Parameters

Name	A name for this connection. This name will be used in the Server Connections drop-down box so that you can distinguish it from other WMS Servers.
URL	URL of the server providing the data. This must be a resolvable host name; the same format as you would use to open a telnet connection or ping a host.

If you need to set up a proxy-server to be able to receive WMS-services from the internet, you can add your proxy-server in the options. Choose menu **Settings** >  **Options** and click on the **Proxy** tab. There you can add your proxy-settings and enable them by setting the **Use proxy for web access**.

Once the new WMS Server connection has been created, it will be preserved for future QGIS sessions.

Table 3 shows some example WMS URLs to get you started. These links were last checked in December 2006, but could change at any time:

An exhaustive list of WMS servers can be found at <http://wms-sites.com>.

Tip 24 ON WMS SERVER URLS

Be sure, when entering in the WMS server URL, that you have the base URL. For example, you shouldn't have fragments such as `request=GetCapabilities` or `version=1.0.0` in your URL.

Table 3: Example Public WMS URLs

Name	URL
Atlas of Canada	http://atlas.gc.ca/cgi-bin/atlaswms_en?
DEMIS	http://www2.demis.nl/wms/wms.asp?wms=WorldMap&
Geoscience Australia	http://www.ga.gov.au/bin/getmap.pl?dataset=national
NASA JPL OnEarth	http://wms.jpl.nasa.gov/wms.cgi?
QGIS Users	http://qgis.org/cgi-bin/mapserv?map=/var/www/maps/main.map&

7.2.3 Loading WMS Layers

Once you have successfully filled in your parameters you can select the **Connect** button to retrieve the capabilities of the selected server. This includes the Image encoding, Layers, Layer Styles, and Projections. Since this is a network operation, the speed of the response depends on the quality of your network connection to the WMS server. While downloading data from the WMS server, the download progress is visualized in the left bottom of the WMS Plugin dialog.

Your screen should now look a bit like Figure 14, which shows the response provided by the NASA JPL OnEarth WMS server.

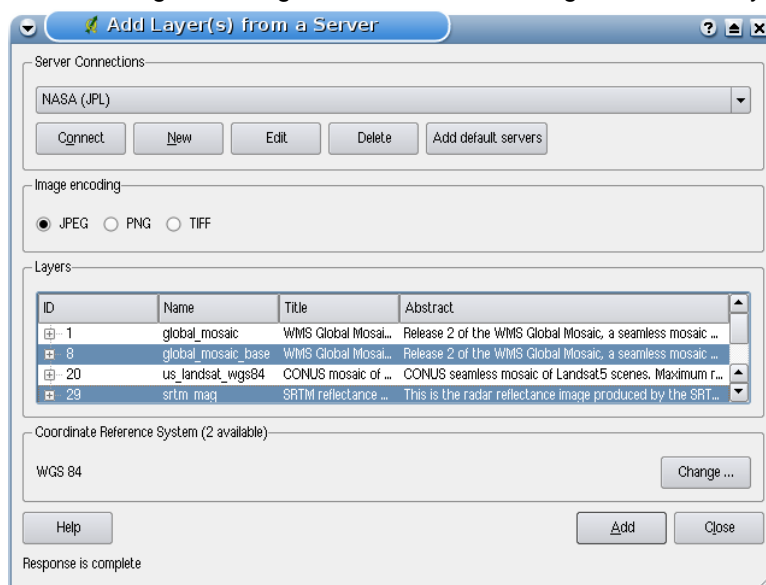
Figure 14: Dialog for adding a WMS server, showing its available layers 🐧

Image Encoding

The **Image encoding** section now lists the formats that are supported by both the client and server. Choose one depending on your image accuracy requirements.

Tip 25 IMAGE ENCODING

You will typically find that a WMS server offers you the choice of JPEG or PNG image encoding. JPEG is a lossy compression format, whereas PNG faithfully reproduces the raw raster data.

Use JPEG if you expect the WMS data to be photographic in nature and/or you don't mind some loss in picture quality. This trade-off typically reduces by 5 times the data transfer requirement compared to PNG. Use PNG if you want precise representations of the original data, and you don't mind the increased data transfer requirements.

Layers

The **Layers** section lists the layers available from the selected WMS server. You may notice that some layers are expandible, this means that the layer can be displayed in a choice of image styles.

You can select several layers at once, but only one image style per layer. When several layers are selected, they will be combined at the WMS Server and transmitted to QGIS in one go.

Tip 26 WMS LAYER ORDERING

In this version of QGIS, WMS layers rendered by a server are overlaid in the order listed in the Layers section,

from top to bottom of the list. If you want to overlay layers in the opposite order, then you can select



Add WMS layer a second time, choose the same server again, and select the second group of layers that you want to overlay the first group.

Transparency

In this version of QGIS, the transparency setting is hard-coded to be always on, where available.

Tip 27 WMS LAYER TRANSPARENCY

The availability of WMS image transparency depends on the image encoding used: PNG and GIF support transparency, whilst JPEG leaves it unsupported.

Coordinate Reference System

A Coordinate Reference System (CRS) is the OGC terminology for a QGIS Projection.


Each WMS Layer can be presented in multiple CRSs, depending on the capability of the WMS server. You may notice that the *x* changes in the *Coordinate Reference System (x available)* header as you select and deselect layers from the **Layers** section.

To choose a CRS, select **Change...** and a screen similar to Figure 17 in Section 8.3 will appear. The main difference with the WMS version of the screen is that only those CRSs supported by the WMS Server will be shown.

Tip 28 WMS PROJECTIONS

For best results, make the WMS layer the first layer you add in the project. This allows the project projection to inherit the CRS you used to render the WMS layer. On-the-fly projection (see Section 8.2) can then be used to fit any subsequent vector layers to the project projection. In this version of QGIS, if you add a WMS layer later, and give it a different CRS to the current project projection, unpredictable results can occur.

7.2.4 Using the Identify Tool

Once you have added a WMS server, and if any layer from a WMS server is queryable, you can then use the  **Identify** tool to select a pixel on the map canvas. A query is made to the WMS server for each selection made.

The results of the query are returned in plain text. The formatting of this text is dependent on the particular WMS server used.

7.2.5 Viewing Properties

Once you have added a WMS server, you can view its properties by right-clicking on it in the legend, and selecting **Properties**.

Metadata Tab

The **Metadata** tab displays a wealth of information about the WMS server, generally collected from the Capabilities statement returned from that server.

Many definitions can be gleaned by reading the WMS standards (5), (6), but here are a few handy definitions:

- **Server Properties**
 - **WMS Version** - The WMS version supported by the server.
 - **Image Formats** - The list of MIME-types the server can respond with when drawing the map. QGIS supports whatever formats the underlying Qt libraries were built with, which is typically at least `image/png` and `image/jpeg`.
 - **Identity Formats** - The list of MIME-types the server can respond with when you use the Identify tool. Currently QGIS supports the `text-plain` type.


- **Layer Properties**

- **Selected** - Whether or not this layer was selected when its server was added to this project.
- **Visible** - Whether or not this layer is selected as visible in the legend. (Not yet used in this version of QGIS.)
- **Can Identify** - Whether or not this layer will return any results when the Identify tool is used on it.
- **Can be Transparent** - Whether or not this layer can be rendered with transparency. This version of QGIS will always use transparency if this is *Yes* and the image encoding supports transparency .
- **Can Zoom In** - Whether or not this layer can be zoomed in by the server. This version of QGIS assumes all WMS layers have this set to *Yes*. Deficient layers may be rendered strangely.
- **Cascade Count** - WMS servers can act as a proxy to other WMS servers to get the raster data for a layer. This entry shows how many times the request for this layer is forwarded to peer WMS servers for a result.
- **Fixed Width, Fixed Height** - Whether or not this layer has fixed source pixel dimensions. This version of QGIS assumes all WMS layers have this set to nothing. Deficient layers may be rendered strangely.
- **WGS 84 Bounding Box** - The bounding box of the layer, in WGS 84 coordinates. Some WMS servers do not set this correctly (e.g. UTM coordinates are used instead). If this is the case, then the initial view of this layer may be rendered with a very “zoomed-out” appearance by QGIS. The WMS webmaster should be informed of this error, which they may know as the WMS XML elements `LatLonBoundingBox`, `EX_GeographicBoundingBox` or the `CRS:84 BoundingBox`.
- **Available in CRS** - The projections that this layer can be rendered in by the WMS server. These are listed in the WMS-native format.
- **Available in style** - The image styles that this layer can be rendered in by the WMS server.

7.2.6 WMS Client Limitations

Not all possible WMS Client functionality had been included in this version of QGIS. Some of the more notable exceptions follow:

Editing WMS Layer Settings

Once you've completed the  **Add WMS layer** procedure, there is no ability to change the settings.

A workaround is to delete the layer completely and start again.

WMS Servers Requiring Authentication


Only public WMS servers are accessible. There is no ability to apply a user name and password combination as an authentication to the WMS server.

Tip 29 ACCESSING SECURED OGC-LAYERS

If you need to access secured layers, you could use InteProxy as a transparent proxy, which does supports several authentication methods. More information can be found at the InteProxy-manual found on the website <http://inteproxy.wald.intevation.org>.

7.3 WFS Client

In QGIS, a WFS layer behaves pretty much like any other vector layer. You can identify and select features and view the attribute table. An exception is that editing is not supported at this time. To start the WFS plugin you need to open **Plugins** > **Plugin Manager...**, activate the **WFS plugin** checkbox and click **OK**.


A new  **Add WFS Layer** icon appears next to the WMS icon. Click on it to open the dialog. In General adding a WFS layer is very similar to the procedure used with WMS. The difference is there are no default servers defined, so we have to add our own.

7.3.1 Loading a WFS Layer

As an example we use the DM Solutions WFS server and display a layer. The URL is:

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?VERSION=1.0.0&SERVICE=wfs&REQUEST=GetCapabilities
```

1. Make sure the WFS plugin is loaded; if not, open the Plugin Manager and load it

2. Click on the  **Add WFS Layer** tool on the plugins toolbar

3. Click on **New**

4. Enter **Name** **DM Solutions** as the name

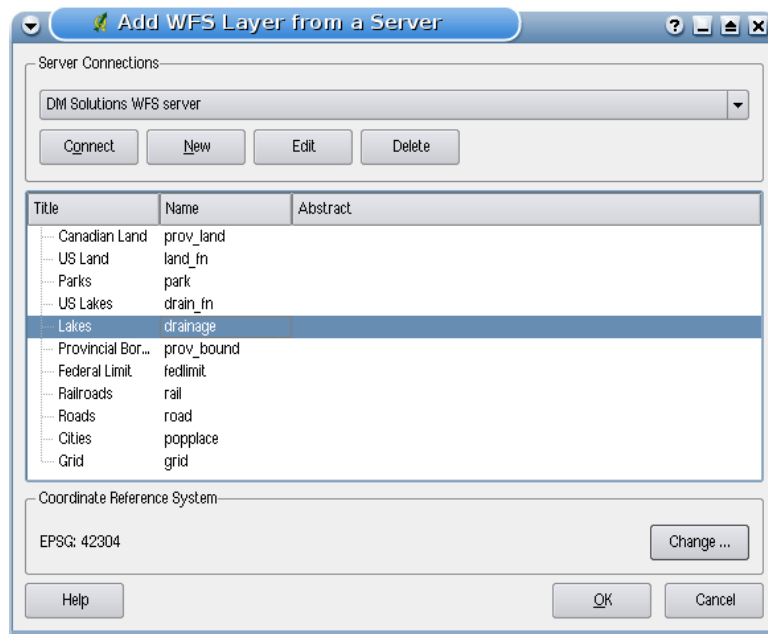
5. Enter the URL (see previous page)

6. Click **OK**

7. Choose **Server Connections** **DM Solutions** from the drop-down box

8. Click **Connect**
9. Wait for the list of layers to be populated
10. Click on the **Canadian Land** layer
11. Click **Add** to add the layer to the map
12. Wait patiently for the features to appear

Figure 15: Adding a WFS layer 



You'll notice the download progress is visualized in the left bottom of the QGIS main window. Once the layer is loaded, you can identify and select a province or two and view the attribute table.

Remember this plugin works best with UMN MapServer WFS servers. It still could be, that you might experience random behavior and crashes. You can look forward to improvements in a future version of the plugin.

Tip 30 FINDING WMS AND WFS SERVERS

You can find additional WMS and WFS servers by using Google or your favorite search engine. There are a number of lists with public URLs, some of them maintained and some not.

8 Working with Projections

QGIS allows users to define a global and project-wide CRS (Coordinate Reference System) for layers without a pre-defined CRS. It also allows the user to define custom coordinate reference systems and supports on-the-fly (OTF) projection of vector layers. All these features allow the user to display layers with different CRS and have them overlay properly.

8.1 Overview of Projection Support

QGIS has support for approximately 2,700 known CRS. Definitions for each of these CRS are stored in a SQLite database that is installed with QGIS. Normally you do not need to manipulate the database directly. In fact, doing so may cause projection support to fail. Custom CRS are stored in a user database. See Section 8.4 for information on managing your custom coordinate reference systems.

The CRS available in QGIS are based on those defined by EPSG and are largely abstracted from the `spatial_references` table in PostGIS version 1.x. The EPSG identifiers are present in the database and can be used to specify a CRS in QGIS.

In order to use OTF projection, your data must contain information about its coordinate reference system or you have to define a global, layer or project-wide CRS. For PostGIS layers QGIS uses the spatial reference identifier that was specified when the layer was created. For data supported by OGR, QGIS relies on the presence of a format specific means of specifying the CRS. In the case of shapefiles, this means a file containing the Well Known Text (WKT) specification of the CRS. The projection file has the same base name as the shapefile and a `prj` extension. For example, a shapefile named `alaska.shp` would have a corresponding projection file named `alaska.prj`.


8.2 Specifying a Projection

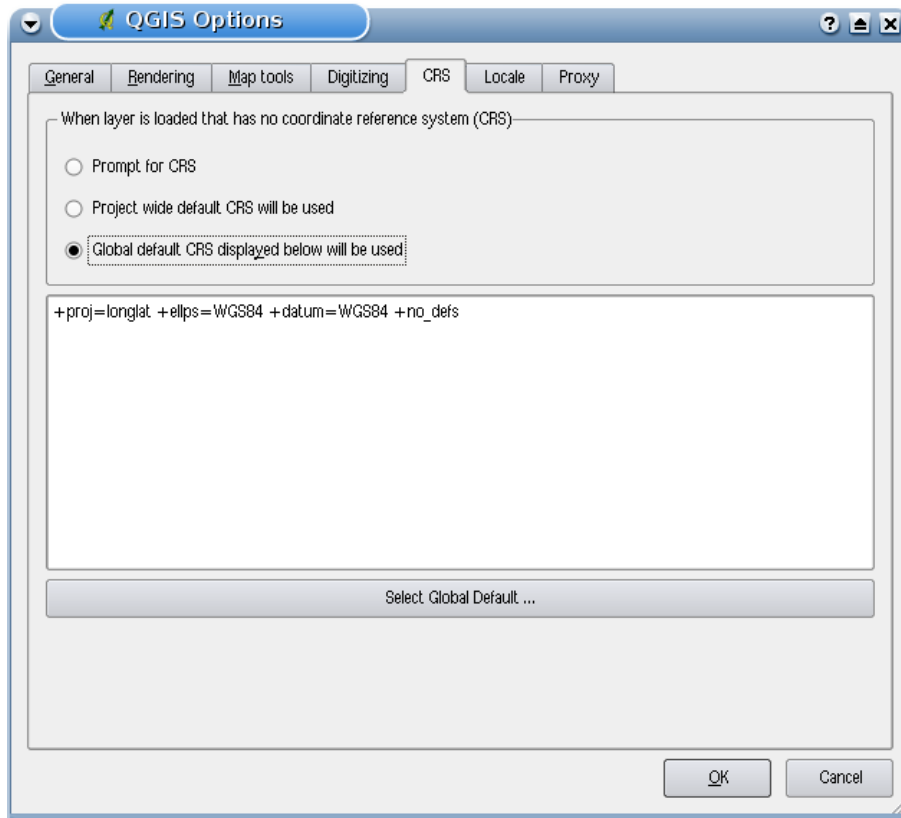
QGIS no longer sets the map CRS to the coordinate reference system of the first layer loaded. When you start a QGIS session with layers that do not have a CRS, you need to control and define the CRS definition for these layers. This can be done globally or project-wide in the **CRS** tab under

Settings > **Options** (See Figure 16).

- Prompt for CRS
- Project wide default CRS will be used
- Global default CRS displayed below will be used

The global default CRS `proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs` comes predefined in QGIS but can of course be changed, and the new definition will be saved for subsequent QGIS sessions.

Figure 16: CRS tab in the QGIS Options Dialog 




If you want to define the coordinate reference system for a certain layer without CRS information, you can also do that in the **General** tab of the raster properties (6.3.4) and vector properties (5.3.1) dialog. If your layer already has a CRS defined, it will be displayed as shown in Figure 6.

8.3 Define On The Fly (OTF) Projection

QGIS does not have OTF projection enabled by default, and this function is currently only supported for vector layers. To use OTF projection, you must open the **Project Properties** dialog, select a CRS and activate the **Enable on the fly projection** checkbox. There are two ways to open the dialog:

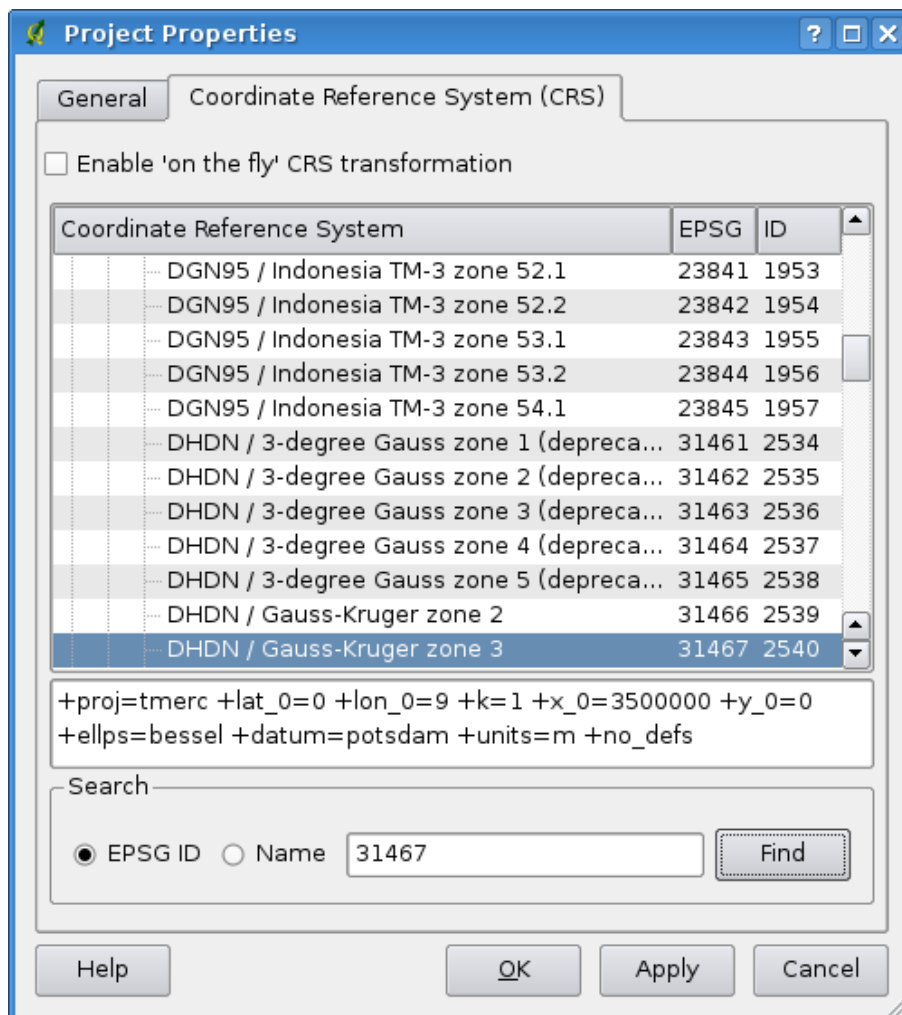
1. Select **Project Properties** from the **Settings** menu.

2. Click on the  projector icon in the lower right-hand corner of the statusbar.

If you have already loaded a layer, and want to enable OTF projection, the best practice is to open the **Coordinate Reference System** tab of the **Project Properties** dialog and find the CRS of the currently loaded layer in the list of CRS, and activate the **Enable on the fly projection** checkbox. All subsequently loaded vector layers will then be OTF projected to the defined CRS.


The **Coordinate Reference System** tab of the **Project Properties** dialog contains four important components as numbered in Figure 17 and described below.

Figure 17: Projection Dialog 


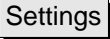




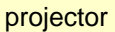
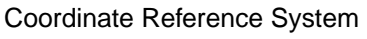
1. **Enable on the fly projection** - this checkbox is used to enable or disable OTF projection. When off, each layer is drawn using the coordinates as read from the data source. When on,

the coordinates in each layer are projected to the coordinate reference system defined for the map canvas.


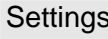
2. **Coordinate Reference System** - this is a list of all CRS supported by QGIS, including Geographic, Projected and Custom coordinate reference systems. To use a CRS, select it from the list by expanding the appropriate node and selecting the CRS. The active CRS is preselected.
3. **Proj4 text** - this is the CRS string used by the Proj4 projection engine. This text is read-only and provided for informational purposes.
4. **Search** - if you know the EPSG identifier or the name for a Coordinate Reference System, you can use the search feature to find it. Enter the identifier and click on .

Tip 31 PROJECT PROPERTIES DIALOG

If you open the  dialog from the  menu, you must click on the

 tab to view the CRS settings. Opening the dialog from the   icon will automatically bring the  tab to the front.

8.4 Custom Coordinate Reference System

If QGIS does not provide the coordinate reference system you need, you can define a custom CRS. To define a CRS, select  **Custom CRS** from the  menu. Custom CRS are stored in your QGIS user database. In addition to your custom CRS, this database also contains your spatial bookmarks and other custom data.

Defining a custom CRS in QGIS requires a good understanding of the Proj.4 projection library. To begin, refer to the Cartographic Projection Procedures for the UNIX Environment - A User's Manual by Gerald I. Evenden, U.S. Geological Survey Open-File Report 90-284, 1990 (available at <ftp://ftp.remotesensing.org/proj/OF90-284.pdf>). This manual describes the use of the `proj.4` and related command line utilities. The cartographic parameters used with `proj.4` are described in the user manual, and are the same as those used by QGIS.

The  dialog requires only two parameters to define a user CRS:

1. a descriptive name and
2. the cartographic parameters in PROJ.4 format.





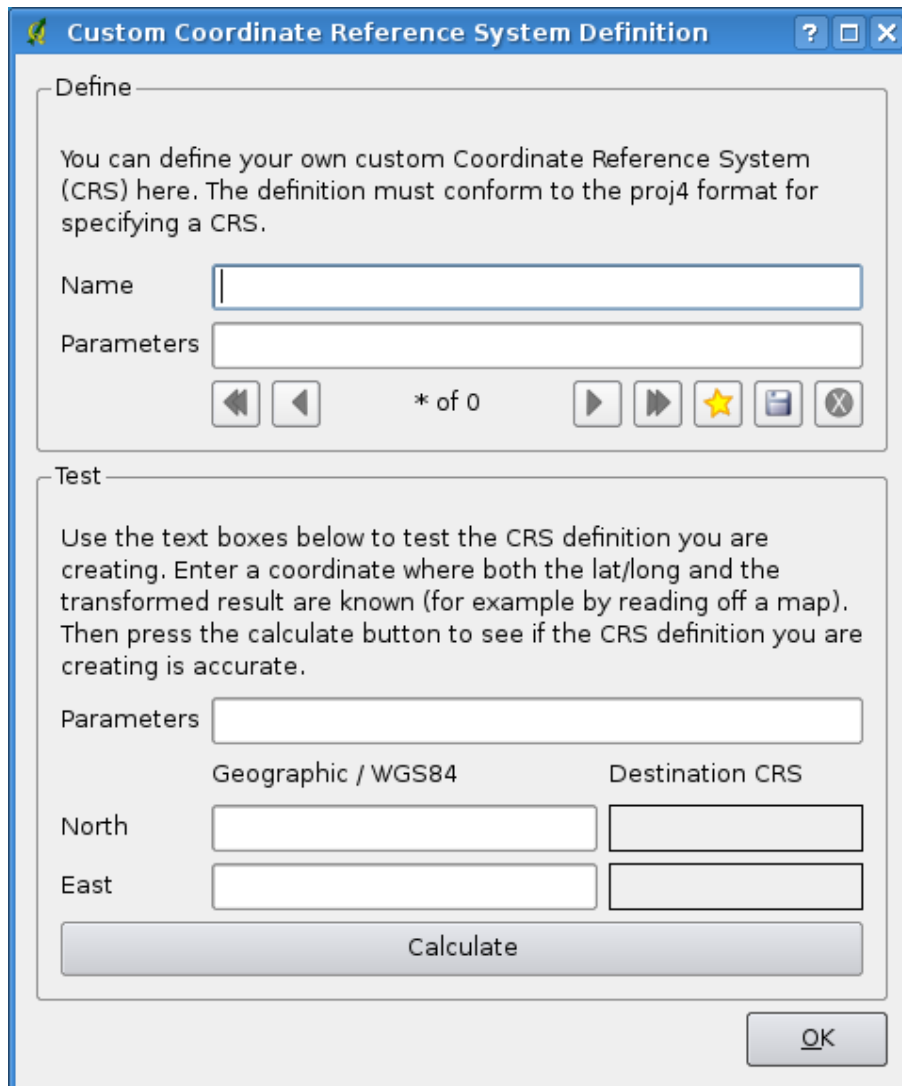
To create a new CRS, click the   button and enter a descriptive name and the CRS parameters. After that you can save your CRS by clicking the button  .

Figure 18: Custom CRS Dialog 


Define

You can define your own custom Coordinate Reference System (CRS) here. The definition must conform to the proj4 format for specifying a CRS.

Name

Parameters

◀ ◀ * of 0 ▶ ▶ ★ 📄 ✕

Test

Use the text boxes below to test the CRS definition you are creating. Enter a coordinate where both the lat/long and the transformed result are known (for example by reading off a map). Then press the calculate button to see if the CRS definition you are creating is accurate.

Parameters

	Geographic / WGS84	Destination CRS
North	<input type="text"/>	<input type="text"/>
East	<input type="text"/>	<input type="text"/>

Calculate

OK











Note that the Parameters must begin with a `+proj=-` block, to represent the new coordinate reference system.

You can test your CRS parameters to see if they give sane results by clicking on the **Calculate** button inside the Test block and pasting your CRS parameters into the Parameters field. Then enter known WGS 84 latitude and longitude values in North and East fields respectively. Click on **Calculate** and compare the results with the known values in your coordinate reference system.

9 GRASS GIS Integration

The GRASS plugin provides access to GRASS GIS (3) databases and functionalities. This includes visualization of GRASS raster and vector layers, digitizing vector layers, editing vector attributes, creating new vector layers and analysing GRASS 2D and 3D data with more than 300 GRASS modules.

In this Section we'll introduce the plugin functionalities and give some examples on managing and working with GRASS data. Following main features are provided with the toolbar menu, when you start the GRASS plugin, as described in Section 9.1:

-  Open mapset
-  New mapset
-  Close mapset
-  Add GRASS vector layer
-  Add GRASS raster layer
-  Create new GRASS vector
-  Edit GRASS vector layer
-  Open GRASS tools
-  Display current GRASS region
-  Edit current GRASS region




9.1 Starting the GRASS plugin

To use GRASS functionalities and/or visualize GRASS vector and raster layers in QGIS, you must select and load the GRASS plugin with the Plugin Manager. Therefore click the menu **Plugins** > **Manage Plugins**, select **GRASS** and click **OK**.

You can now start loading raster and vector layers from an existing GRASS LOCATION (see Section 9.2). Or you create a new GRASS LOCATION with QGIS (see Section 9.3.1) and import some raster and vector data (see Section 9.4) for further analysis with the GRASS Toolbox (see Section 9.9).

9.2 Loading GRASS raster and vector layers

With the GRASS plugin, you can load vector or raster layers using the appropriate button on the toolbar menu. As an example we use the QGIS alaska dataset (see Section 3.2). It includes a small sample GRASS LOCATION with 3 vector layers and 1 raster elevation map.

1. Create a new folder `grassdata`, download the QGIS alaska dataset `qgis_sample_data.zip` from <http://download.osgeo.org/qgis/data/> and unzip the file into `grassdata`.
2. Start QGIS.
3. If not already done in a previous QGIS session, load the GRASS plugin clicking on **Plugins** > **Manage Plugins** and selecting **GRASS**. The GRASS toolbar appears on the toolbar menu.
4. In the GRASS toolbar, click the  **Open mapset** icon to bring up the MAPSET wizard.
5. For `Gisdbase` browse and select or enter the path to the newly created folder `grassdata`.
6. You should now be able to select the LOCATION `alaska` and the MAPSET `demo`.
7. Click **OK**. Notice that some previously disabled tools in the GRASS toolbar are now enabled.
8. Click on  **Add GRASS raster layer**, choose the map name `gtopo30` and click **OK**. The elevation layer will be visualized.
9. Click on  **Add GRASS vector layer**, choose the map name `alaska` and click **OK**. The alaska boundary vector layer will be overlaid on top of the `gtopo30` map. You can now adapt the layer properties as described in chapter 5.3, e.g. change opacity, fill and outline color.
10. Also load the other two vector layers `rivers` and `airports` and adapt their properties.

As you see, it is very simple to load GRASS raster and vector layers in QGIS. See following Sections for editing GRASS data and creating a new LOCATION. More sample GRASS LOCATIONs are available at the GRASS website at <http://grass.osgeo.org/download/data.php>.

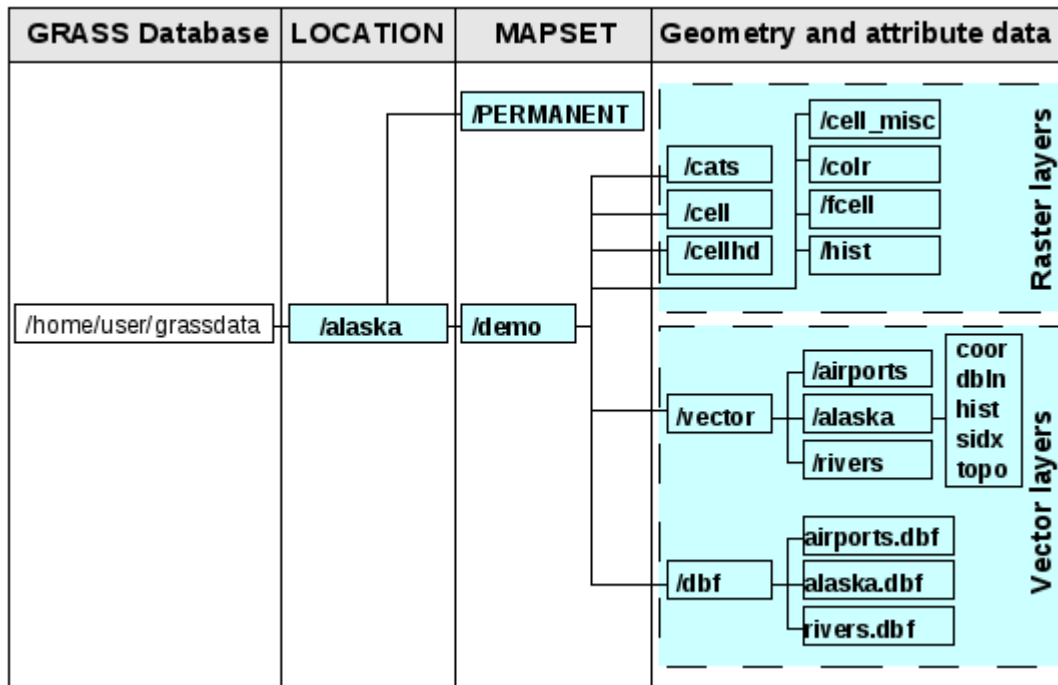
Tip 32 GRASS DATA LOADING

If you have problems loading data or QGIS terminates abnormally, check to make sure you have loaded the GRASS plugin properly as described in Section 9.1.

9.3 GRASS LOCATION and MAPSET

GRASS data are stored in a directory referred to as GISDBASE. This directory often called `grassdata`, must be created before you start working with the GRASS plugin in QGIS. Within this directory, the GRASS GIS data are organized by projects stored in subdirectories called `LOCATION`. Each `LOCATION` is defined by its coordinate system, map projection and geographical boundaries. Each `LOCATION` can have several `MAPSET`s (subdirectories of the `LOCATION`) that are used to subdivide the project into different topics, subregions, or as workspaces for individual team members (Neteler & Mitasova 2008 (2)). In order to analyze vector and raster layers with GRASS modules, you must import them into a GRASS `LOCATION`.⁴

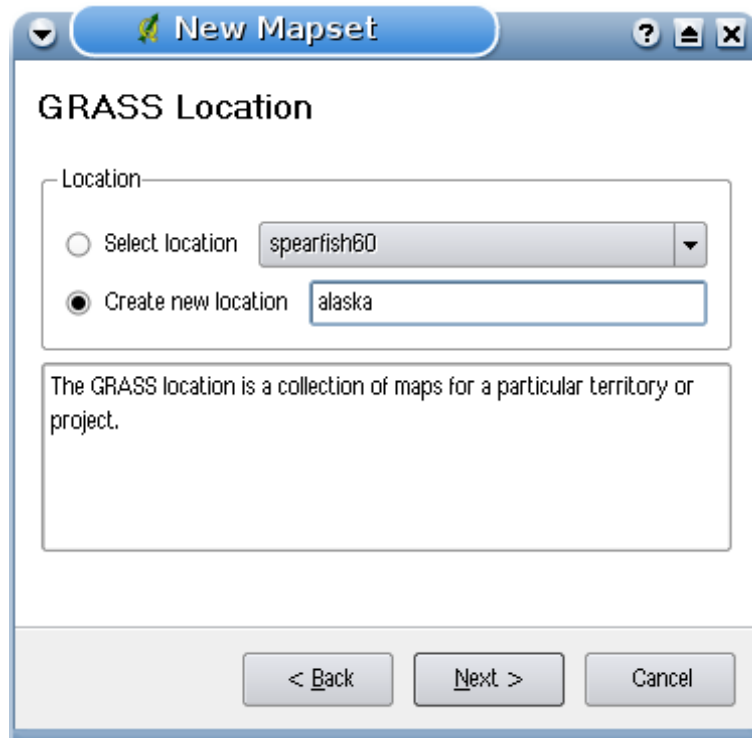
Figure 19: GRASS data in the alaska LOCATION (adapted from Neteler & Mitasova 2008 (2))





9.3.1 Creating a new GRASS LOCATION

As an an example you find the instructions how the sample GRASS `LOCATION` `alaska`, which is projected in Albers Equal Area projection with unit meter was created for the QGIS sample dataset. This sample GRASS `LOCATION` `alaska` will be used for all examples and exercises in the following GRASS GIS related chapters. It is useful to download and install the dataset on your computer (3.2).

⁴This is not strictly true - with the GRASS modules `r.external` and `v.external` you can create read-only links to external GDAL/OGR-supported data sets without importing them. But because this is not the usual way for beginners to

Figure 20: Creating a new GRASS LOCATION or a new MAPSET in QGIS 

1. Start QGIS and make sure the GRASS plugin is loaded
2. Visualize the `alaska.shp` Shapefile (see Section 5.1.1) from the QGIS alaska dataset 3.2.
3. In the GRASS toolbar, click on the  **Open mapset** icon to bring up the MAPSET wizard.
4. Select an existing GRASS database (GISDBASE) folder `grassdata` or create one for the new LOCATION using a file manager on your computer. Then click **Next**.
5. We can use this wizard to create a new MAPSET within an existing LOCATION (see Section 9.3.2) or to create a new LOCATION altogether. Click on the radio button Create new location (see Figure 20).
6. Enter a name for the LOCATION - we used `alaska` and click **Next**.
7. Define the projection by clicking on the radio button Projection to enable the projection list
8. We are using Albers Equal Area Alaska (meters) projection. Since we happen to know that it is represented by the EPSG ID 5000, we enter it in the search box. (Note: If you want to repeat this process for another LOCATION and projection and haven't memorized the EPSG ID, click on the  **projector** icon in the lower right-hand corner of the status bar (see Section 8.3)).


work with GRASS, this functionality will not be described here.

9. Click **Find** to select the projection
10. Click **Next**
11. To define the default region, we have to enter the LOCATION bounds in north, south, east, and west direction. Here we simply click on the button **Set current QGIS extent**, to apply the extend of the loaded layer `alaska.shp` as the GRASS default region extend.
12. Click **Next**
13. We also need to define a MAPSET within our new LOCATION. You can name it whatever you like - we used `demo`.⁵
14. Check out the summary to make sure it's correct and click **Finish**
15. The new LOCATION `alaska` and two MAPSETs `demo` and `PERMANENT` are created. The currently opened working set is MAPSET `demo`, as you defined.
16. Notice that some of the tools in the GRASS toolbar that were disabled are now enabled.

If that seemed like a lot of steps, it's really not all that bad and a very quick way to create a LOCATION. The LOCATION `alaska` is now ready for data import (see Section 9.4). You can also use the already existing vector and raster data in the sample GRASS LOCATION `alaska` included in the QGIS `alaska` dataset 3.2 and move on to Section 9.5.

9.3.2 Adding a new MAPSET

A user has only write access to a GRASS MAPSET he created. This means, besides access to his own MAPSET, each user can also read maps in other user's MAPSETs, but he can modify or remove only the maps in his own MAPSET. All MAPSETs include a WIND file that stores the current boundary coordinate values and the currently selected raster resolution (Neteler & Mitasova 2008 (2), see Section 9.8).



1. Start QGIS and make sure the GRASS plugin is loaded
2. In the GRASS toolbar, click on the  **Open mapset** icon to bring up the MAPSET wizard.
3. Select the GRASS database (GISDBASE) folder `grassdata` with the LOCATION `alaska`, where we want to add a further MAPSET, called `test`.
4. Click **Next**.
5. We can use this wizard to create a new MAPSET within an existing LOCATION or to create a new LOCATION altogether. Click on the radio button **Select location** (see Figure 20) and click **Next**.
6. Enter the name `test` for the new MAPSET. Below in the wizard you see a list of existing MAPSETs and its owners.

⁵When creating a new LOCATION, GRASS automatically creates a special MAPSET called `PERMANENT` designed to store the core data for the project, its default spatial extend and coordinate system definitions (Neteler & Mitasova 2008 (2)).

- Click **Next** , check out the summary to make sure it's all correct and click **Finish**

9.4 Importing data into a GRASS LOCATION

This Section gives an example how to import raster and vector data into the `alaska` GRASS LOCATION provided by the QGIS `alaska` dataset. Therefore we use a landcover raster map `landcover.tif` and a vector polygon Shape `lakes.shp` from the QGIS `alaska` dataset 3.2.

- Start QGIS and make sure the GRASS plugin is loaded.
- In the GRASS toolbar, click the  **Open MAPSET** icon to bring up the MAPSET wizard.
- Select as GRASS database the folder `grassdata` in the QGIS `alaska` dataset, as LOCATION `alaska`, as MAPSET `demo` and click **OK** .
- Now click the  **Open GRASS tools** icon. The GRASS Toolbox (see Section 9.9) dialog appears.
- To import the raster map `landcover.tif`, click the module `r.in.gdal` in the **Modules Tree** tab. This GRASS module allows to import GDAL supported raster files into a GRASS LOCATION. The module dialog for `r.in.gdal` appears.
- Browse to the folder `raster` in the QGIS `alaska` dataset and select the file `landcover.tif`.
- As raster output name define `landcover_grass` and click **Run** . In the **Output** tab you see the currently running GRASS command `r.in.gdal -o input=/path/to/landcover.tif output=landcover_grass`.
- When it says **Successfully finished** click **View output** . The `landcover_grass` raster layer is now imported into GRASS and will be visualized in the QGIS canvas.
- To import the vector shape `lakes.shp`, click the module `v.in.ogr` in the **Modules Tree** tab. This GRASS module allows to import OGR supported vector files into a GRASS LOCATION. The module dialog for `v.in.ogr` appears.
- Browse to the folder `vmap0_shapefiles` in the QGIS `alaska` dataset and select the file `lakes.shp` as OGR file.
- As vector output name define `lakes_grass` and click **Run** . You don't have to care about the other options in this example. In the **Output** tab you see the currently running GRASS command `v.in.ogr -o dsn=/path/to/lakes.shp output=lakes_grass`.
- When it says **Successfully finished** click **View output** . The `lakes_grass` vector layer is now imported into GRASS and will be visualized in the QGIS canvas.

9.5 The GRASS vector data model

It is important to understand the GRASS vector data model prior to digitizing. In general, GRASS uses a topological vector model. This means that areas are not represented as closed polygons, but by one or more boundaries. A boundary between two adjacent areas is digitized only once, and it is shared by both areas. Boundaries must be connected without gaps. An area is identified (labeled) by the centroid of the area.

Besides boundaries and centroids, a vector map can also contain points and lines. All these geometry elements can be mixed in one vector and will be represented in different so called 'layers' inside one GRASS vector map. So in GRASS a layer is not a vector or raster map but a level inside a vector layer. This is important to distinguish carefully.⁶

It is possible to store more 'layers' in one vector dataset. For example, fields, forests and lakes can be stored in one vector. Adjacent forest and lake can share the same boundary, but they have separate attribute tables. It is also possible to attach attributes to boundaries. For example, the boundary between lake and forest is a road, so it can have a different attribute table.

The 'layer' of the feature is defined by 'layer' inside GRASS. 'Layer' is the number which defines if there are more than one layer inside the dataset, e.g. if the geometry is forest or lake. For now, it can be only a number, in the future GRASS will also support names as fields in the user interface.


Attributes can be stored inside the GRASS LOCATION as DBase or SQLITE3 or in external database tables, for example PostgreSQL, MySQL, Oracle, etc.

Attributes in database tables are linked to geometry elements using a 'category' value. 'Category' (key, ID) is an integer attached to geometry primitives, and it is used as the link to one key column in the database table.

Tip 33 LEARNING THE GRASS VECTOR MODEL

The best way to learn the GRASS vector model and its capabilities is to download one of the many GRASS tutorials where the vector model is described more deeply. See <http://grass.osgeo.org/gdp/manuals.php> for more information, books and tutorials in several languages.

9.6 Creating a new GRASS vector layer

To create a new GRASS vector layer with the GRASS plugin click the  **Create new GRASS vector** toolbar icon. Enter a name in the text box and you can start digitizing point, line or polygon geometries, following the procedure described in Section 9.7.


⁶Although it is possible to mix geometry elements, it is unusual and even in GRASS only used in special cases such as vector network analysis. Normally you should prefer to store different geometry elements in different layers.

In GRASS it is possible to organize all sort of geometry types (point, line and area) in one layer, because GRASS uses a topological vector model, so you don't need to select the geometry type when creating a new GRASS vector. This is different from Shapefile creation with QGIS, because Shapefiles use the Simple Feature vector model (see Section 5.4.4).

Tip 34 CREATING AN ATTRIBUTE TABLE FOR A NEW GRASS VECTOR LAYER

If you want to assign attributes to your digitized geometry features, make sure to create an attribute table with columns before you start digitizing (see Figure 25).

9.7 Digitizing and editing a GRASS vector layer

The digitizing tools for GRASS vector layers are accessed using the  **Edit GRASS vector layer** icon on the toolbar. Make sure you have loaded a GRASS vector and it is the selected layer in the legend before clicking on the edit tool. Figure 22 shows the GRASS edit dialog that is displayed when you click on the edit tool. The tools and settings are discussed in the following sections.

Tip 35 DIGITIZING POLYGONES IN GRASS

If you want to create a polygon in GRASS, you first digitize the boundary of the polygon, setting the mode to `No` category. Then you add a centroid (label point) into the closed boundary, setting the mode to `Next not used`. The reason is, that a topological vector model links attribute information of a polygon always to the centroid and not to the boundary.

Toolbar

In Figure 21 you see the GRASS digitizing toolbar icons provided by the GRASS plugin. Table 4 explains the available functionalities.

Figure 21: GRASS Digitizing Toolbar 















Category Tab

The **Category** tab allows you to define the way in which the category values will be assigned to a new geometry element.

- **Mode:** what category value shall be applied to new geometry elements.
 - Next not used - apply next not yet used category value to geometry element.

Table 4: GRASS Digitizing Tools

Icon	Tool	Purpose
	New Point	Digitize new point
	New Line	Digitize new line (finish by selecting new tool)
	New Boundary	Digitize new boundary (finish by selecting new tool)
	New Centroid	Digitize new centroid (label existing area)
	Move vertex	Move one vertex of existing line or boundary and identify new position
	Add vertex	Add a new vertex to existing line
	Delete vertex	Delete vertex from existing line (confirm selected vertex by another click)
	Move element	Move selected boundary, line, point or centroid and click on new position
	Split line	Split an existing line to 2 parts
	Delete element	Delete existing boundary, line, point or centroid (confirm selected element by another click)
	Edit attributes	Edit attributes of selected element (note that one element can represent more features, see above)
	Close	Close session and save current status (rebuilds topology afterwards)

- Manual entry - manually define the category value for the geometry element in the 'Category'-entry field.
- No category - Do not apply a category value to the geometry element. This is e.g. used for area boundaries, because the category values are connected via the centroid.
- **Category** - A number (ID) is attached to each digitized geometry element. It is used to connect each geometry element with its attributes.
- **Field (layer)** - Each geometry element can be connected with several attribute tables using different GRASS geometry layers. Default layer number is 1.

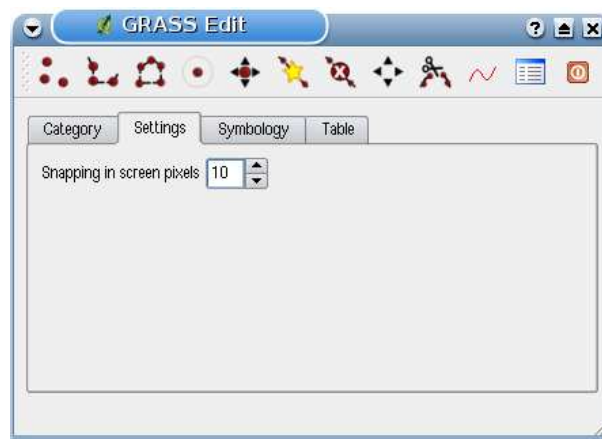
Tip 36 CREATING AN ADDITIONAL GRASS 'LAYER' WITH QGIS

If you would like to add more layers to your dataset, just add a new number in the 'Field (layer)' entry box and press return. In the Table tab you can create your new table connected to your new layer.

Figure 22: GRASS Digitizing Category Tab 

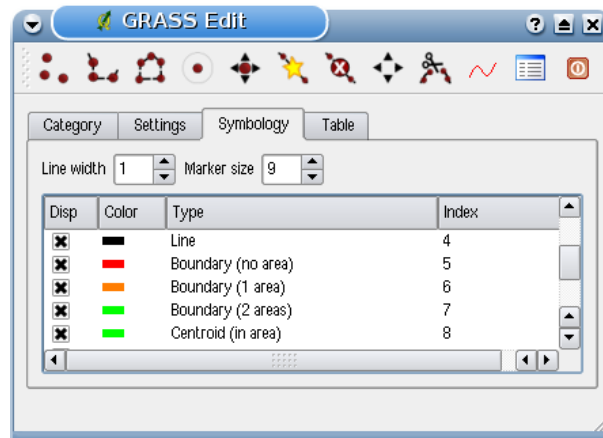
Settings Tab

The **Settings** tab allows you to set the snapping in screen pixels. The threshold defines at what distance new points or line ends are snapped to existing nodes. This helps to prevent gaps or dangles between boundaries. The default is set to 10 pixels.

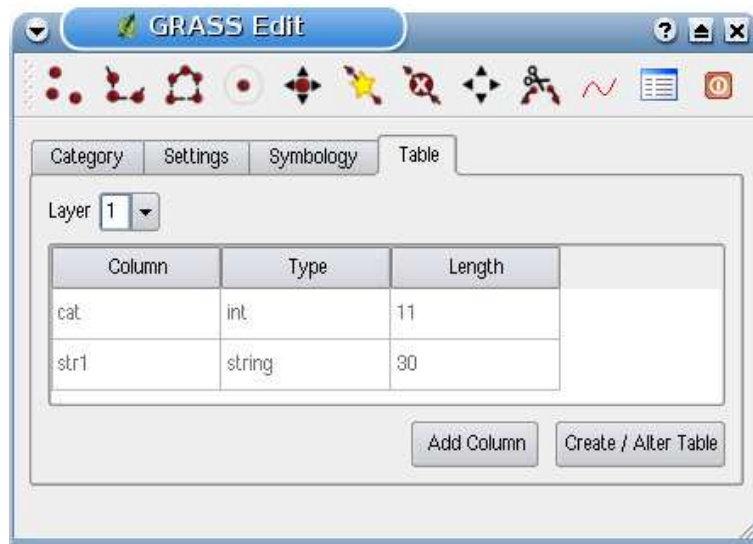
Figure 23: GRASS Digitizing Settings Tab 

Symbology Tab

The **Symbology** tab allows you to view and set symbology and color settings for various geometry types and their topological status (e.g. closed / opened boundary).

Figure 24: GRASS Digitizing Symbology Tab **Table Tab**

The **Table** tab provides information about the database table for a given 'layer'. Here you can add new columns to an existing attribute table, or create a new database table for a new GRASS vector layer (see Section 9.6).

Figure 25: GRASS Digitizing Table Tab **Tip 37 GRASS EDIT PERMISSIONS**

You must be the owner of the GRASS MAPSET you want to edit. It is impossible to edit data layers in a MAPSET that is not yours, even if you have write permissions.


9.8 The GRASS region tool

The region definition (setting a spatial working window) in GRASS is important for working with raster layers. Vector analysis is per default not limited to any defined region definitions. All newly-created rasters will have the spatial extension and resolution of the currently defined GRASS region, regardless of their original extension and resolution. The current GRASS region is stored in the `$LOCATION/$MAPSET/WIND` file, and it defines north, south, east and west bounds, number of columns and rows, horizontal and vertical spatial resolution.

It is possible to switch on/off the visualization of the GRASS region in the QGIS canvas using the



Display current GRASS region button. .

With the  **Edit current GRASS region** icon you can open a dialog to change the current region and the symbology of the GRASS region rectangle in the QGIS canvas. Type in the new region bounds and resolution and click **OK**. It also allows to select a new region interactively with your mouse on the QGIS canvas. Therefore click with the left mouse button in the QGIS canvas, open a rectangle, close it using the left mouse button again and click **OK**. The GRASS module `g.region` provide a lot more parameters to define an appropriate region extend and resolution for your raster analysis. You can use these parameters with the GRASS Toolbox, described in Section 9.9.

9.9 The GRASS toolbox




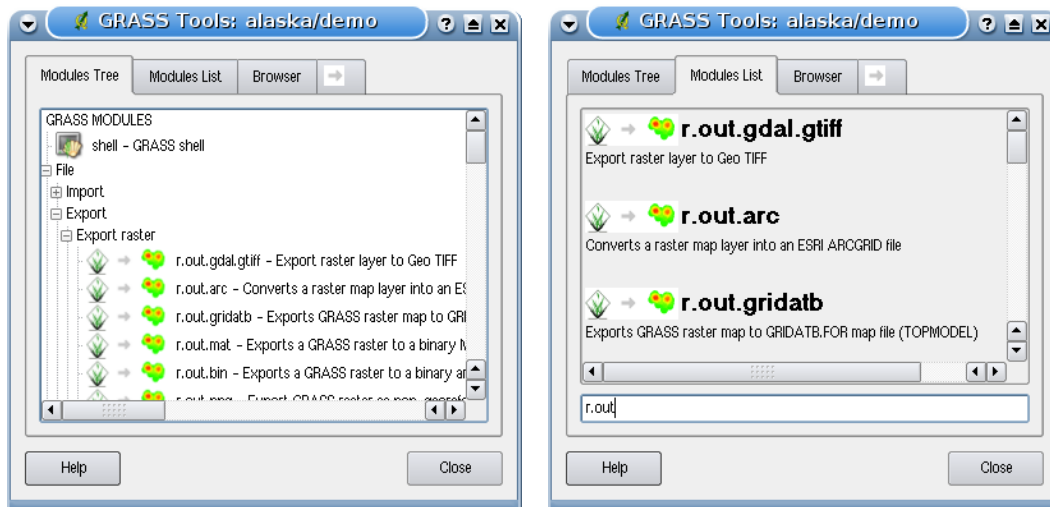
The **Open GRASS Tools** box provides GRASS module functionalities to work with data inside a selected GRASS LOCATION and MAPSET. To use the GRASS toolbox you need to open a LOCATION and MAPSET where you have write-permission (usually granted, if you created the MAPSET). This is necessary, because new raster or vector layers created during analysis need to be written to the currently selected LOCATION and MAPSET.

9.9.1 Working with GRASS modules

The GRASS Shell inside the GRASS Toolbox provides access to almost all (more than 300) GRASS modules in command line modus. To offer a more user friendly working environment, about 200 of the available GRASS modules and functionalities are also provided by graphical dialogs. These dialogs are grouped in thematic blocks, but are searchable as well. You find a complete list of GRASS modules available in QGIS version 1.0.0 in appendix B. It is also possible to customize the GRASS Toolbox content. It is described in Section 9.9.3.

As shown in Figure 26, you can look for the appropriate GRASS module using the thematically

Figure 26: GRASS Toolbox and searchable Modules List 



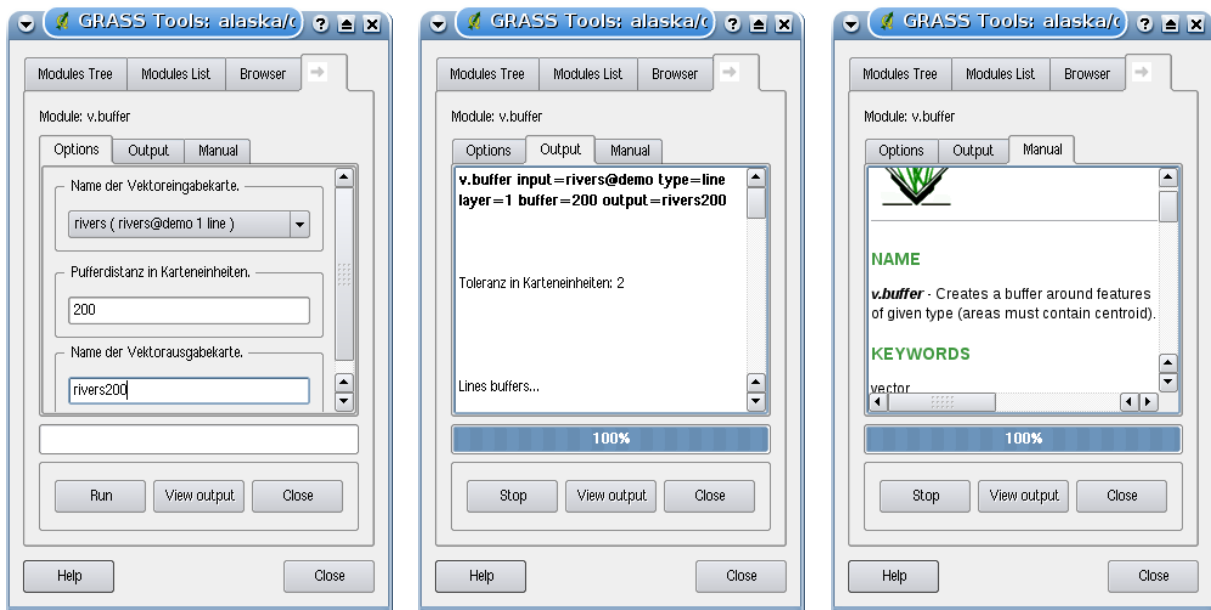
(a) Modules Tree

(b) Searchable Modules List

grouped **Modules Tree** or the searchable **Modules List** tab.

Clicking on a graphical module icon a new tab will be added to the toolbox dialog providing three new sub-tabs **Options**, **Output** and **Manual**. In Figure 27 you see an example for the GRASS module `v.buffer`.

Figure 27: GRASS Toolbox Module Dialogs 



(a) Module Options

(b) Modules Output

(c) Module Manual

Options

The **Options** tab provides a simplified module dialog where you can usually select a raster or vector layer visualized in the QGIS canvas and enter further module specific parameters to run the module. The provided module parameters are often not complete to keep the dialog clear. If you want to use further module parameters and flags, you need to start the GRASS Shell and run the module in the command line.

Output

The **Output** tab provides information about the output status of the module. When you click the **Run** button, the module switches to the **Output** tab and you see information about the analysis process. If all works well, you will finally see a `Successfully finished` message.

Manual

The **Manual** tab shows the HTML help page of the GRASS module. You can use it to check further module parameters and flags or to get a deeper knowledge about the purpose of the module. At the end of each module manual page you see further links to the `Main Help index`, the `Thematic index` and the `Full index`. These links provide the same information as if you use the module `g.manual`

Tip 38 DISPLAY RESULTS IMMEDIATELY

If you want to display your calculation results immediately in your map canvas, you can use the 'View Output' button at the bottom of the module tab.

9.9.2 Working with the GRASS LOCATION browser

Another useful feature inside the GRASS Toolbox is the GRASS LOCATION browser. In Figure 28 you can see the current working LOCATION with its MAPSETS.

In the left browser windows you can browse through all MAPSETS inside the current LOCATION. The right browser window shows some meta information for selected raster or vector layers, e.g. resolution, bounding box, data source, connected attribute table for vector data and a command history.

The toolbar inside the **Browser** tab offers following tools to manage the selected LOCATION:




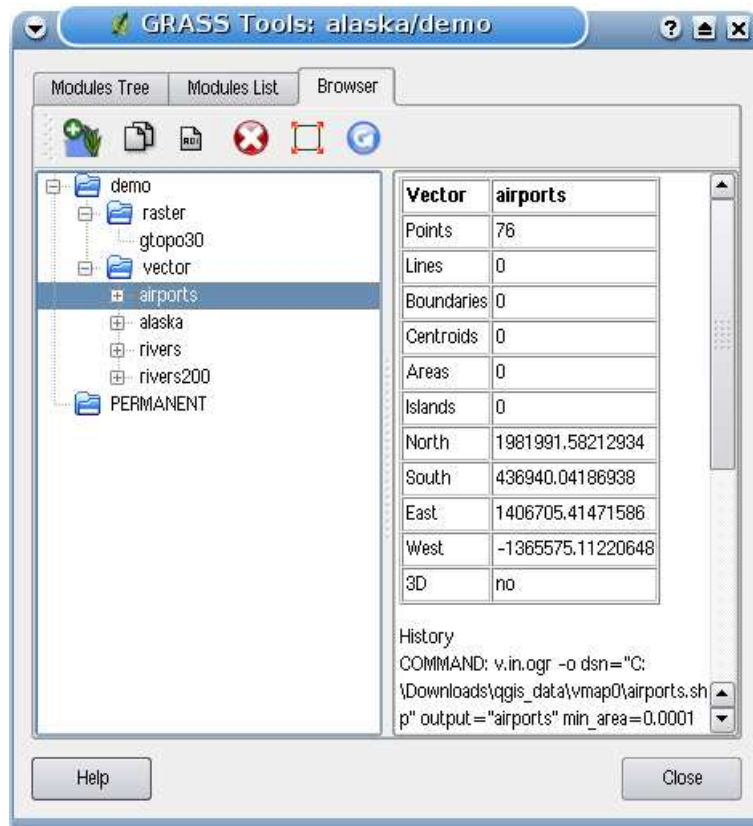





-  Add selected map to canvas
-  Copy selected map
-  Rename selected map

Figure 28: GRASS LOCATION browser 

-  Delete selected map
-  Set current region to selected map
-  Refresh browser window

The  Rename selected map and  Delete selected map only work with maps inside your currently selected MAPSET. All other tools also work with raster and vector layers in another MAPSET.

9.9.3 Customizing the GRASS Toolbox

Nearly all GRASS modules can be added to the GRASS toolbox. A XML interface is provided to parse the pretty simple XML files which configures the modules appearance and parameters inside the toolbox.

A sample XML file for generating the module `v.buffer` (`v.buffer.qgm`) looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE qgisgrassmodule SYSTEM "http://mrcc.com/qgisgrassmodule.dtd">

<qgisgrassmodule label="Vector buffer" module="v.buffer">
  <option key="input" typeoption="type" layeroption="layer" />
  <option key="buffer"/>
  <option key="output" />
</qgisgrassmodule>
```





















The parser reads this definition and creates a new tab inside the toolbox when you select the module. A more detailed description for adding new modules, changing the modules group, etc. can be found on the QGIS wiki at



http://wiki.qgis.org/qgiswiki/Adding_New_Tools_to_the_GRASS_Toolbox.

10 Print Composer


The print composer provides growing layout and printing capabilities. It allows you to add elements such as the QGIS map canvas, legend, scalebar, images, and text labels. You can size, group and position each element and adjust the properties to create your layout. The result can be printed (also to Postscript and PDF), exported to image formats or to SVG.⁷ See a list of tools in table 5:

Table 5: Print Composer Tools

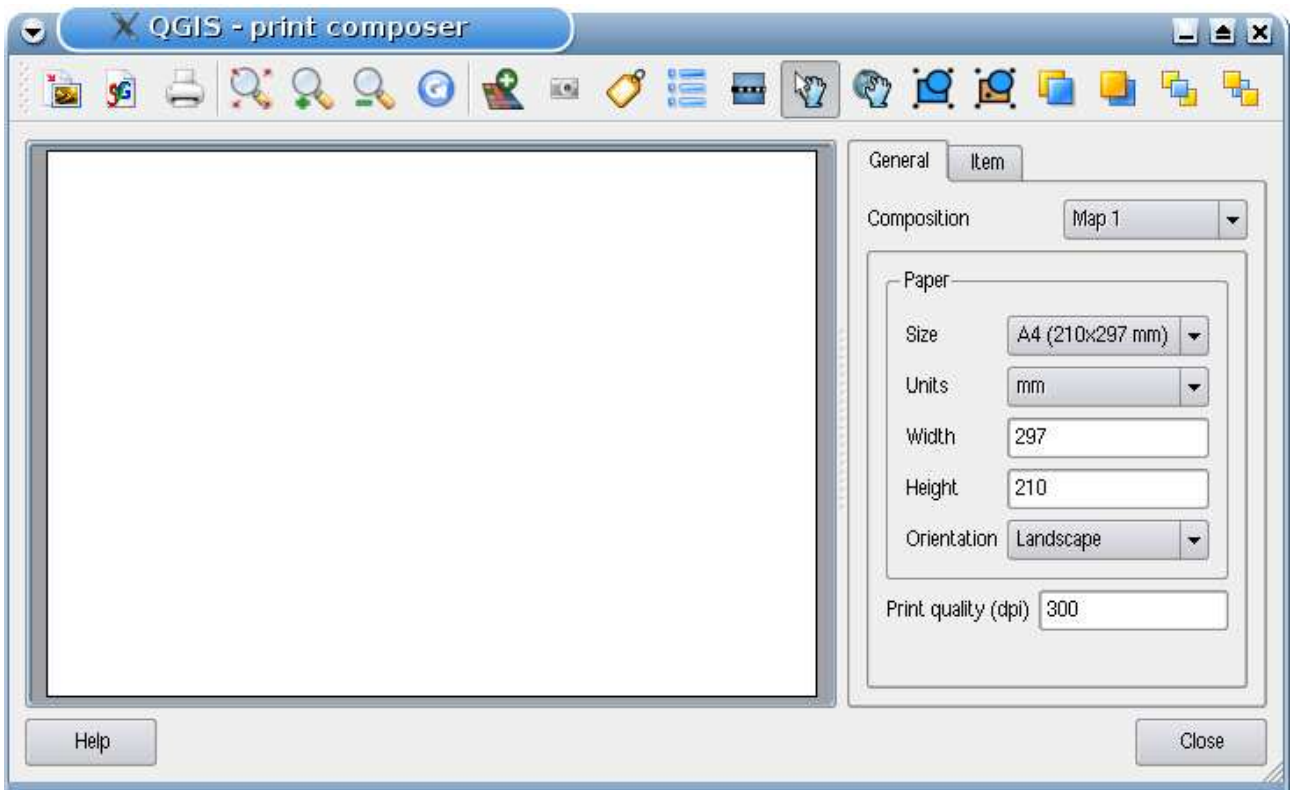
Icon	Purpose	Icon	Purpose
	Export to an image format		Export print composition to SVG
	Print or export as PDF or Postscript		Zoom to full extent
	Zoom in		Zoom out
	Refresh view		Add new map from QGIS map canvas
	Add Image to print composition		Add label to print composition
	Add new legend to print composition		Add new scalebar to print composition
	Select/Move item in print composition		Move content within an item
	Group items of print composition		Ungroup items of print composition
	Raise selected items in print composition		Lower selected items in print composition
	Move selected items to top		Move selected items to bottom

To access the print composer, click on the  **Print** button in the toolbar or choose **File** >  **Print Composer**.


10.1 Using Print Composer

Before you start to work with the print composer, you need to load some raster and vector layers in the QGIS map canvas and adapt their properties to suite your own convenience. After everything is rendered and symbolized to your liking you click the  **Print Composer** icon.

⁷Export to SVG supported, but it is not working properly with some recent QT4 versions. You should try and check individual on your system

Figure 29: Print Composer 

Opening the print composer provides you with a blank canvas to which you can add the current QGIS map canvas, legend, scalebar, images and text. Figure 29 shows the initial view of the print composer before any elements are added. The print composer provides two tabs:

- The **General** tab allows you to set paper size, orientation, and the print quality for the output file in dpi.
- The **Item** tab displays the properties for the selected map element. Click the  **Select/Move item** icon to select an element (e.g. legend, scalebar or label) on the canvas. Then click the Item tab and customize the settings for the selected element.

You can add multiple elements to the composer. It is also possible to have more than one map view or legend or scalebar in the print composer canvas. Each element has its own properties and in the case of the map, its own extent.

10.1.1 Adding a current QGIS map canvas to the Print Composer



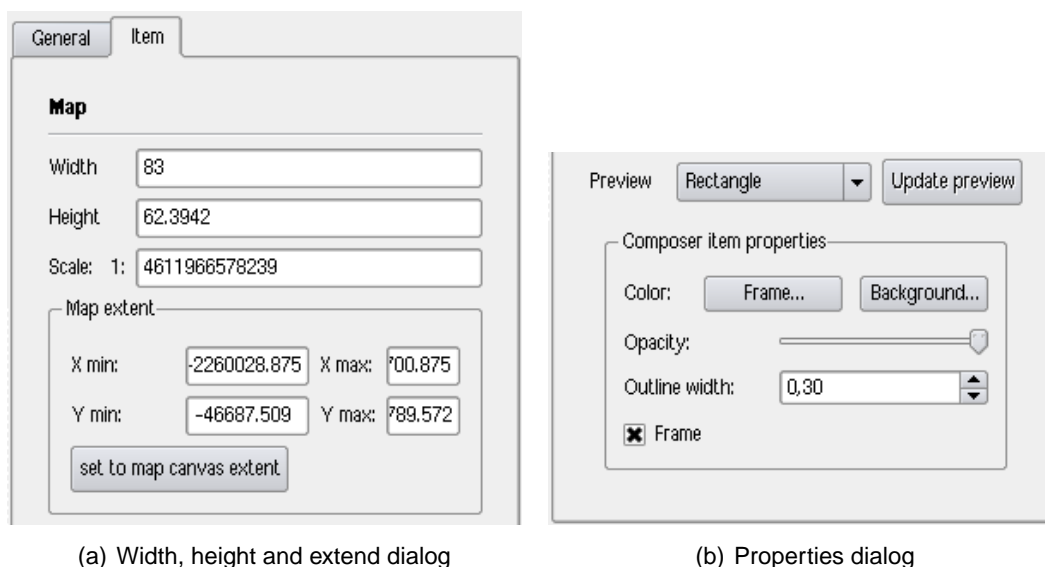

To add the QGIS map canvas, click on the  **Add new map from QGIS map canvas** button in the print composer toolbar and drag a rectangle on the composer canvas with the left mouse button to add the map. You will see an empty box with a "Map will be printed here" message. To display the current map, choose **Preview** **Cache**  in the map **Item** tab.


Figure 30: Print Composer map item tab content 



(a) Width, height and extend dialog

(b) Properties dialog

You can resize the map later by clicking on the  **Select/Move item** button, selecting the element, and dragging one of the blue handles in the corner of the map. With the map selected, you can now adapt more properties in the map **Item** tab. Resize the map item specifying the width and height or the scale. Define the map extend using Y and X min/max values or clicking the **set to map canvas extent** button. Update the map preview and select, whether to see a preview from cache or an empty rectangle with a "Map will be printed here" message. Define colors and outline width for the element frame, set a background color and opacity for the map canvas. And you can also select or unselect to display an element frame with the **Frame** checkbox (see Figure 30). If you change the view on the QGIS map canvas by zooming or panning or changing vector or raster properties, you can update the print composer view selecting the map element in the print composer and clicking the **Update Preview** button in the map **Item** tab (see Figure 30).

To move layers within the map element select the map element, click the  **Move item content** icon and move the layers within the map element frame with the left mouse button.

Tip 39 SAVING A PRINT COMPOSER LAYOUT

If you want to save the current state of a print composer session, click on **File** > **Save Project As** to save the state of your workspace including the state of the current print composer session. It is planned but currently not possible to save print composer templates itself.

10.1.2 Adding other elements to the Print Composer

Besides adding a current QGIS map canvas to the Print Composer, it is also possible to add, move and customize legend, scalebar, images and label elements.

Label and images




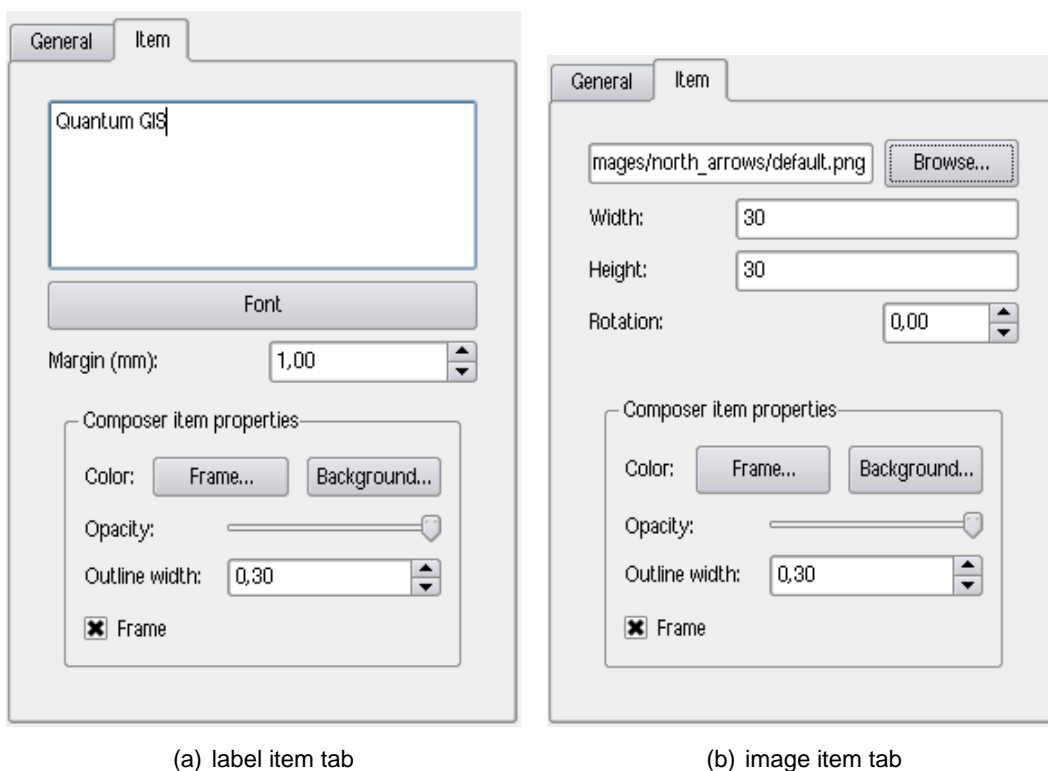
To add a label or an image, click the  **Add label** or  **Add image** icon and place the element with the left mouse button on the print composer canvas.

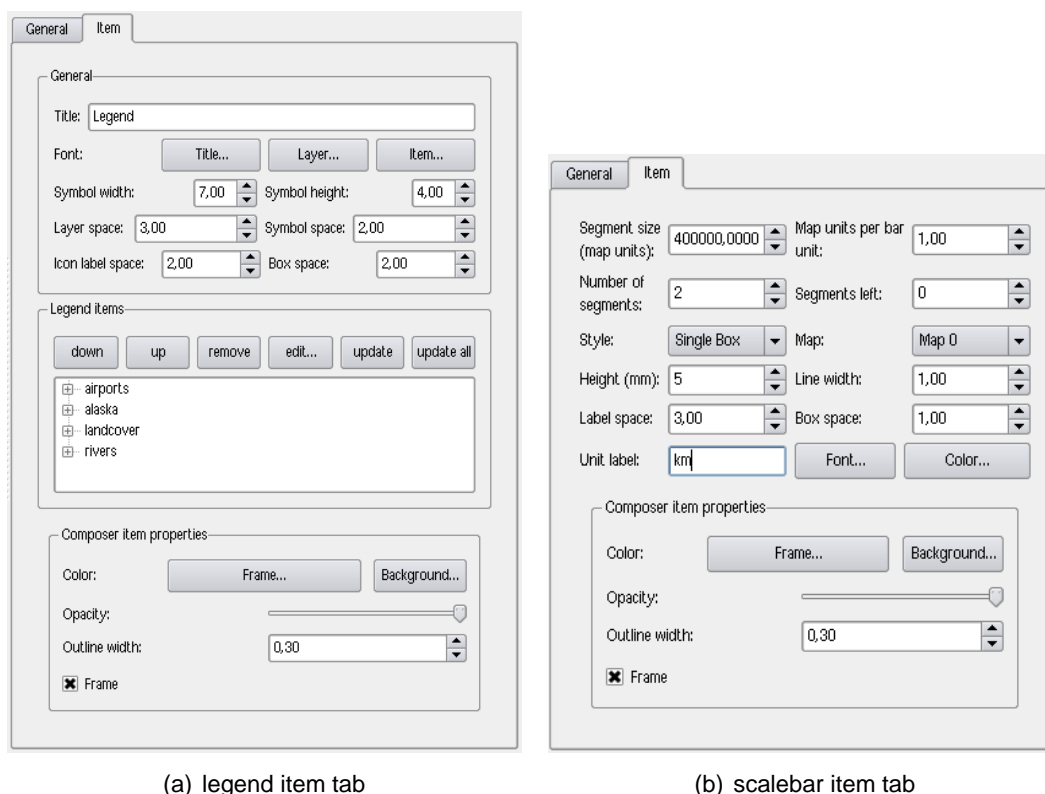
Figure 31: Customize print composer label and images 



Legend and scalebar

To add a map legend or a scalebar, click the  **Add new legend** or  **Add new scalebar** icon and place the element with the left mouse button on the print composer canvas.

Figure 32: Customize print composer legend and scalebar 







(a) legend item tab

(b) scalebar item tab

10.1.3 Navigation tools

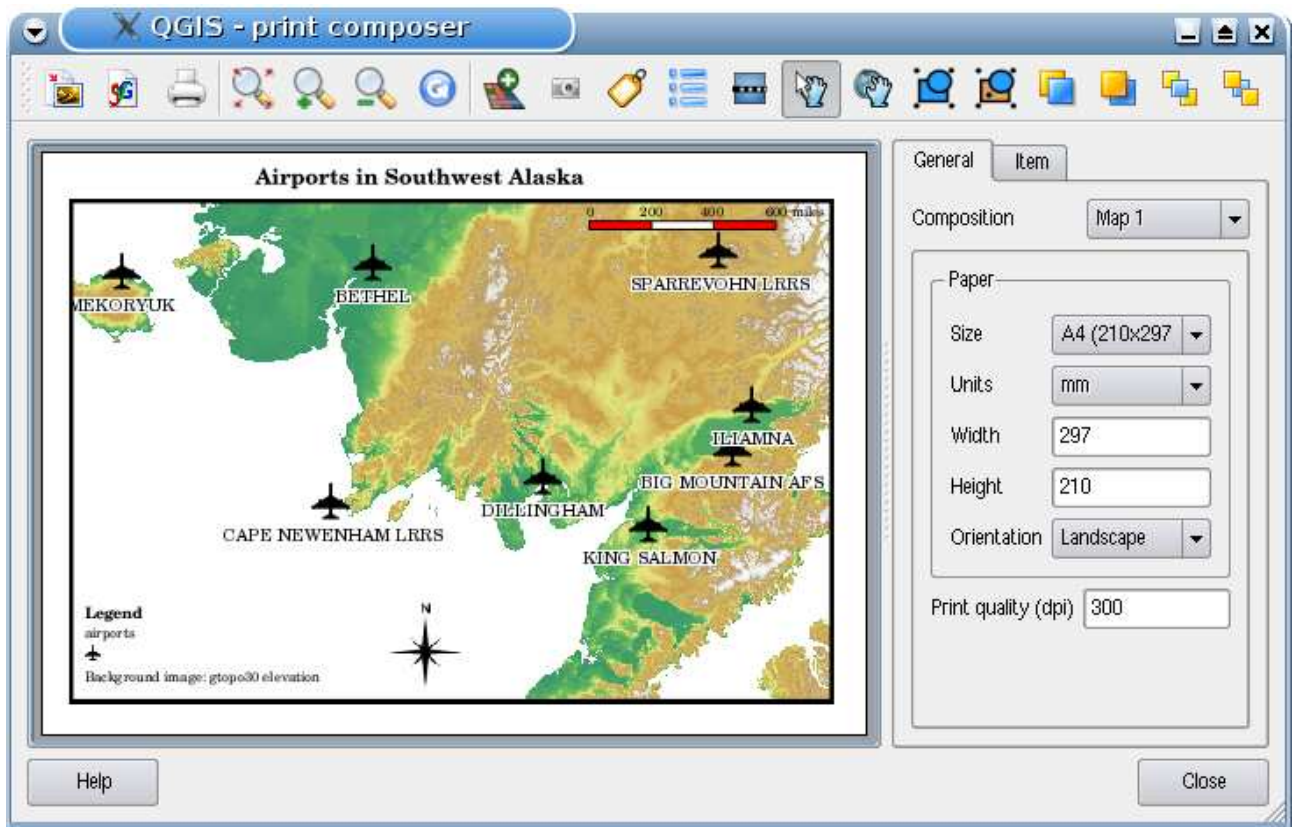
For map navigation the print composer provides 4 general tools:

-  **Zoom in**,
-  **Zoom out**,
-  **Zoom to full extent** and
-  **Refresh the view**, if you find the view in an inconsistent state.




10.1.4 Creating Output

Figure 33 shows the print composer with an example print layout including each type of map element described in the sections above.

Figure 33: Print Composer with map view, legend, scalebar, and text added 



The print composer allows you to create several output formats and it is possible to define the resolution (print quality) and paper size:

- The  **Print** icon allows to print the layout to a connected printer or as PDF or Postscript file depending on installed printer drivers.
- The  **Export as image** icon exports the composer canvas in several image formats such as PNG, BPM, TIF, JPG, ...
- The  **Export as SVG** icon saves the print composer canvas as a SVG (Scalable Vector Graphic). **Note:** Currently the SVG output is very basic. This is not a QGIS problem, but a problem of the underlying Qt library. This will hopefully be sorted out in future versions.

11 QGIS Plugins

QGIS has been designed with a plugin architecture. This allows new features/functions to be easily added to the application. Many of the features in QGIS are actually implemented as **core** or **external** plugins.

- **Core Plugins** are maintained by the QGIS Development Team and are automatically part of every QGIS distribution. They are written in one of two languages: C++ or Python. More information about core plugins are provided in Section 12.
- **External Plugins** are currently all written in Python. They are stored in external repositories and maintained by the individual author. They can be added to QGIS using the core plugin called `Plugin Installer`. More information about external plugins are provided in Section 13.

11.1 Managing Plugins

Managing plugins in general means loading or unloading them using the `Plugin Manager` plugin. External plugins need to be first installed using the `Plugin Installer` plugin.

11.1.1 Loading a QGIS Core Plugin

Loading a QGIS Core Plugin is provided in the main menu `Plugins` > `Manage Plugins...`.

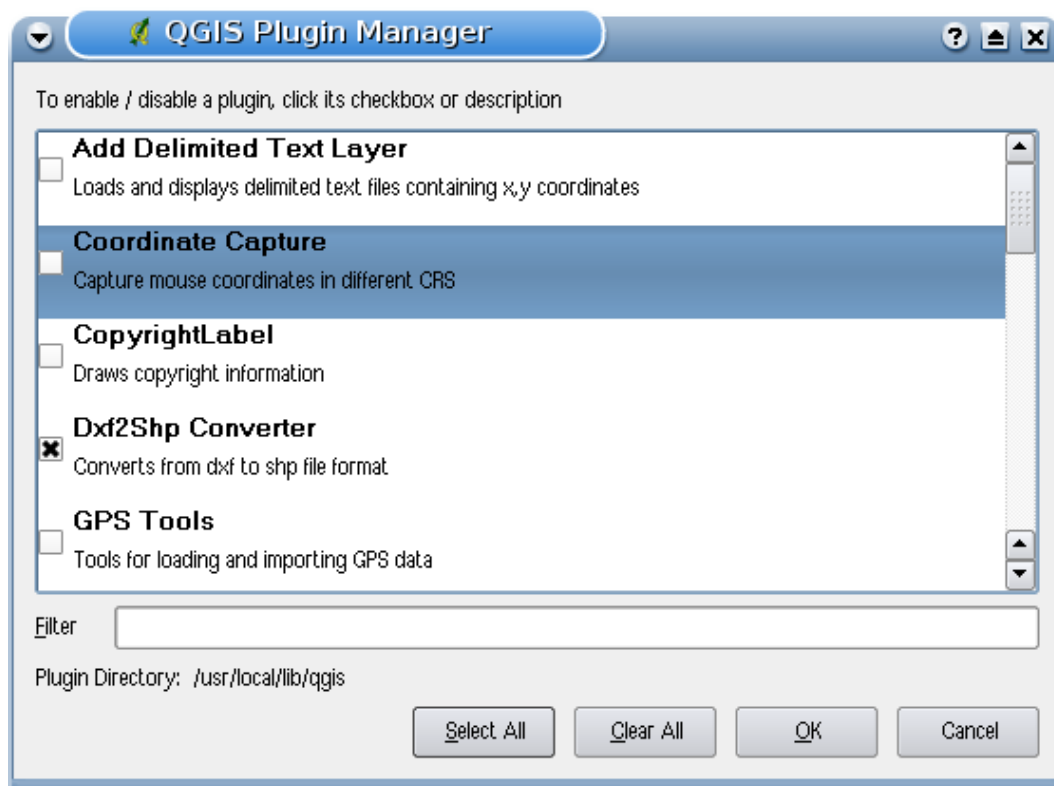
The `Plugin Manager` lists all the available plugins and their status (loaded or unloaded). All available means all core plugins and all external plugins you added using `Plugin Installer` plugin (see Section 13). Figure 34 shows the `Plugin Manager` dialog. Loaded plugins are "remembered" when you exit the application and restored the next time you run QGIS.

Tip 40 CRASHING PLUGINS

If you find that QGIS crashes on startup, a plugin may be at fault. You can stop all plugins from loading by editing your stored settings file (see 4.7 for location). Locate the plugins settings and change all the plugin values to false to prevent them from loading. 🐧 For example, to prevent the `Delimited text` plugin from loading, the entry in `$HOME/.config/QuantumGIS/qgis.conf` on Linux should look like this: `Add Delimited Text Layer=false`. Do this for each plugin in the `[Plugins]` section. You can then start QGIS and add the plugins one at a time from the `Plugin Manger` to determine which is causing the problem.

11.1.2 Loading an external QGIS Plugin

To be able to integrate external plugins into QGIS you first need to load the `Plugin Installer` plugin as described in Section 11.1.1. Then you can load external QGIS python plugin in two steps:

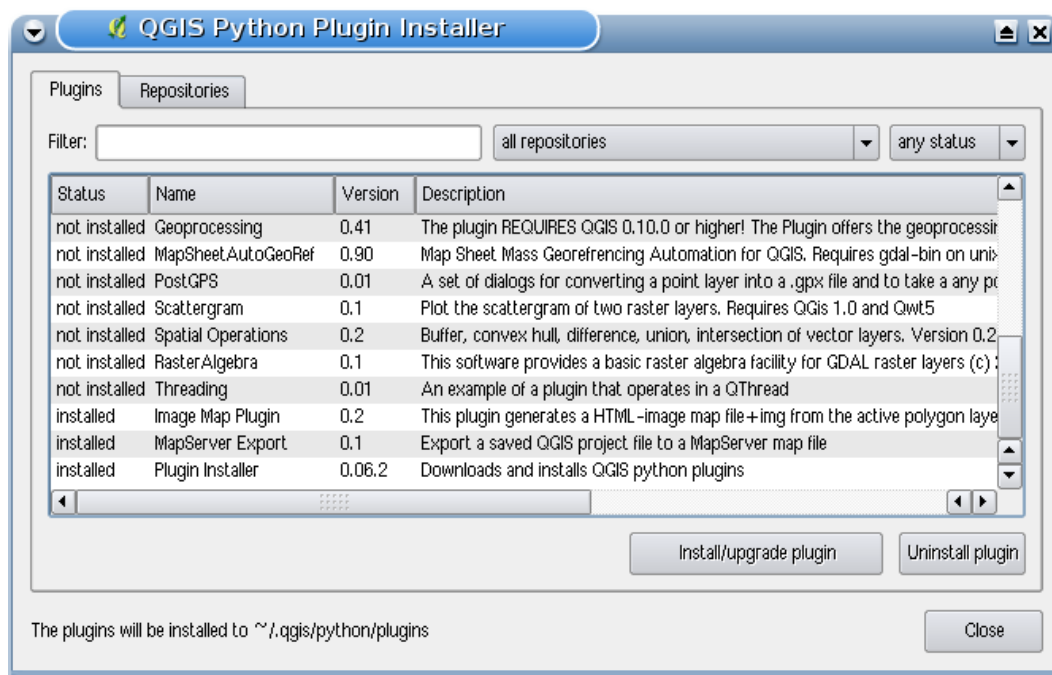
Figure 34: Plugin Manager 

1. Download an external plugin from a repository using the Plugin Installer (Section 11.1.3). The new external plugin will be integrated into the list of available plugins in the Plugin Manager.
2. Load the plugin using the Plugin Manager.

11.1.3 Using the QGIS Python Plugin Installer

In order to download and install an external Python plugin, click the menu **Plugins** > **Fetch Python Plugins...**. The Plugin Installer window will appear (figure 35) with the tab **Plugins**, containing the list of all Python plugins available in remote repositories as well as installed ones. Each plugin can be either:


- **not installed** - it means the plugin is available in the repository, but is not installed yet. In order to install, select it from the list and click the **Install plugin** button.
- **new** - the same as before but the plugin is seen for the first time.
- **installed** - the plugin is installed. If it's also available in any repository the **Reinstall plugin**


Figure 35: Installing external python plugins 


button is enabled. But if the available version is older than the installed one, the **Downgrade plugin** button appears instead.

- **upgradeable** - the plugin is installed, but there is an updated version available. The **Upgrade plugin** button is enabled.
- **invalid** - the plugin is installed, but is unworkable. The reason is explained in the plugin description.

Plugins tab

To install a plugin, select it from the list and click the **Install plugin** button. The plugin is installed in its own directory, e.g. for  under `$HOME/.qgis/python/plugins` and is only visible for the user who has installed it. See a list of other OS specific subdirectory used for plugins in Section 15.3. If the installation is successful, a confirmation message will appear. Then you need go to the **Plugins** > **Manage Plugins...** and load the installed plugin.

If the installation fails, the reason is displayed. The most often troubles are related to connection errors and missing Python modules. In the former case you'll probably need to wait some minutes or hours, in the latter one you need to install the missing modules in your operating system prior to using the plugin.  For Linux, most required modules should be available in a package manager.

 For install instructions in Windows visit the module home page. If you use a proxy, you may need to configure it under the menu **Settings** > **Options** on the **Proxy** tab.

The **Uninstall plugin** button is enabled only if the selected plugin is installed and it's not a core plugin. Note that if you have installed an update of a core plugin, you can still uninstall this update with the **Uninstall plugin** and revert to the version shipped within Quantum GIS install package. This one cannot be uninstalled.

Repositories tab

The second tab **Repositories** contains a list of plugin repositories available for the Plugin Installer. By default, only the QGIS Official Repository is used. You can add some user-contributed repositories, including the central QGIS Contributed Repository and a few author repositories by clicking the **Add 3rd party repositories** button. Those repositories contain a huge number of more or less useful plugins but please note that they aren't maintained by the QGIS Development Team and we can't take any responsibility for them. You can also manage the repository list manually, that is add, remove and edit the entries. Temporary disabling a particular repository is possible clicking the **Edit...** button.

The **Check for updates on startup** checkbox makes QGIS looking for plugin updates and news. If it's enabled, all repositories listed and enabled on the **Repositories** tab are checked whenever the program is starting. If a new plugin or an update for one of installed plugins is available, a clickable notification appears in the Status Bar. If the checkbox is disabled, looking for updates and news is performed only when Plugin Installer is being launched from the menu.

In case of some internet connection problems a *Looking for new plugins...* indicator in the Status Bar may stay visible during whole QGIS session and cause a program crash when exiting. In this case please disable the checkbox.

11.2 Data Providers


















Data Providers are "special" plugins that provides access to a data store. By default, QGIS supports PostGIS layers and disk-based data stores supported by the GDAL/OGR library (Appendix A.1). A Data Provider plugin extends the ability of QGIS to use other data sources.

Data Provider plugins are registered automatically by QGIS at startup. They are not managed by the Plugin Manager but used behind the scenes when a data type is added as a layer in QGIS.

12 Using QGIS Core Plugins

QGIS currently contains 17 core plugins that can be loaded using the Plugin Manager. Table 6 lists each of the core plugins along with a description of their purpose and the toolbar-icon.⁸

Table 6: QGIS Core Plugins

Icon	Plugin	Description
	Add Delimited Text Layer	Loads and displays delimited text files containing x,y coordinates
	Coordinate Capture	Capture mouse coordinate in different CRS
	Copyright Label	Draws a copyright label with information
	DXF2Shape Converter	Converts from DXF to SHP file format
	GPS Tools	Tools for loading and importing GPS data
	GRASS	Activates the mighty GRASS Toolbox
	Georeferencer	Adding projection info to Rasterfiles
	Graticule Creator	Create a latitude/longitude grid and save as a shapefile
	Interpolation plugin	Interpolation on base of vertices of a vector layer
	MapServer Export Plugin	Export a saved QGIS project file to a MapServer map file
	North Arrow	Displays a north arrow overlaid onto the map
	OGR Layer Converter	Translate vector layers between OGR supported formats
	Plugin Installer	Downloads and installs QGIS python plugins
	SPIT	Shapefile to PostgreSQL/PostGIS Import Tool
	Qucik Print	Quickly print a map with minimal effort
	Scalebar	Draws a scale bar
	WFS	Load and display WFS layer

Tip 41 PLUGINS SETTINGS SAVED TO PROJECT

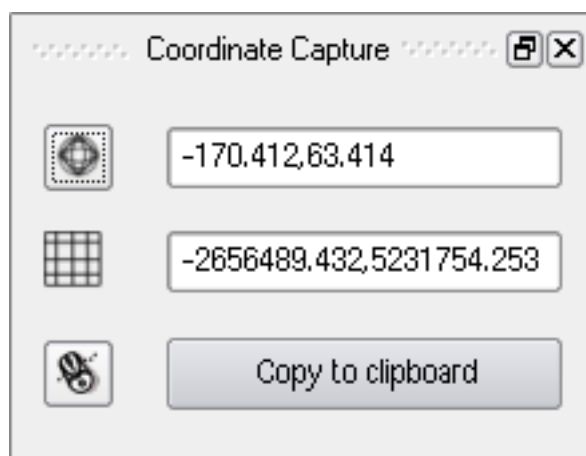
When you save a .qgs project, any changes you have made to NorthArrow, ScaleBar and Copyright plugins will be saved in the project and restored next time you load the project.






⁸The MapServer Export Plugin and the Plugin Installer Plugin are external Python Plugins, but they are part of the QGIS sources and automatically loaded and selectable inside the QGIS Plugin Manager.

12.1 Coordinate Capture Plugin

The coordinate capture plugin is easy to use and provides the capability to display coordinates on the map canvas for two selected Coordinate Reference Systems (CRS). You can click a certain point and copy the coordinates to the clipboard or you use the mouse tracking functionality

Figure 36: Coordinate Capture Plugin 



1. Start QGIS, select  **Project Properties** from the **Settings** menu and click on the **Projection** tab. As an alternative you can also click on the  **projector** icon in the lower right-hand corner of the statusbar.
2. Click on the **Enable on the fly projection** checkbox and select the projected coordinate system "NAD27/Alaska Albers" with EPSG 2964 (see also Section 8).
3. Load the `alaska.shp` vector layer from the qgis sample dataset.
4. Load the coordinate capture plugin in the Plugin Manager (see Section 11.1.1) and click on the  **Coordinate Capture** icon. The coordinate capture dialog appears as shown in Figure 36.
5. Click on the  **Click to the select the CRS to use for coordinate display** icon and select Geographic Coordinate System WGS84 (EPSG 4326).
6. You can now click anywhere on the map canvas and the plugin will show the "NAD27/Alaska Albers" and WGS84 coordinates for your selected points as shown in Figure 36.
7. To enable mouse coordinate tracking click the  **mouse tracking** icon.
8. You can also copy selected coordinates to the clipboard.

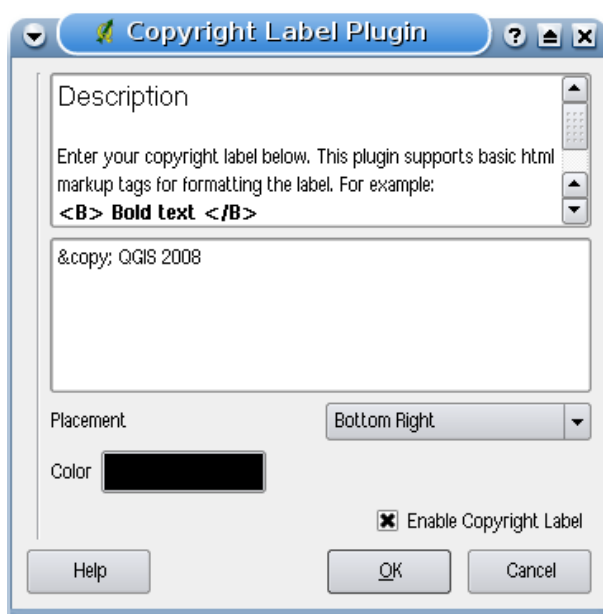
12.2 Decorations Plugins


The Decorations Plugins includes the Copyright Label Plugin, the North Arrow Plugin and the Scale Bar Plugin. They are used to “decorate” the map by adding cartographic elements.

12.2.1 Copyright Label Plugin

The title of this plugin is a bit misleading - you can add any random text to the map.

Figure 37: Copyright Label Plugin 



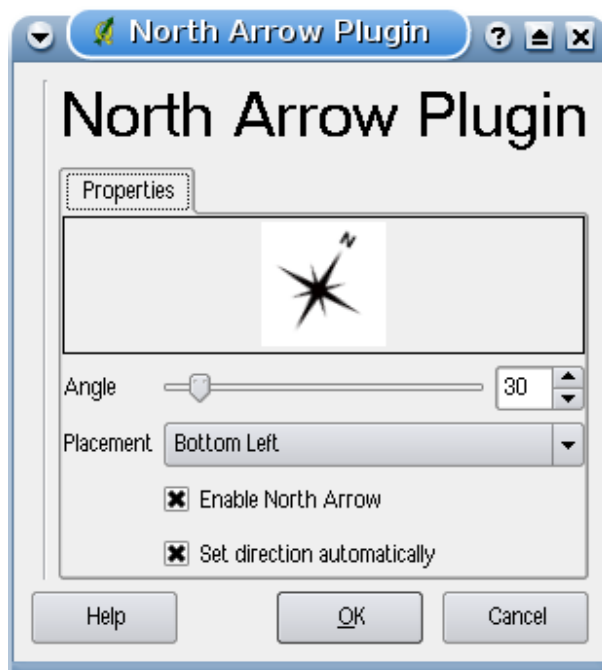
1. Make sure the plugin is loaded
2. Click on **Plugins** > **Decorations** > **Copyright Label** or use the  **Copyright Label** button from the Toolbar.
3. Enter the text you want to place on the map. You can use HTML as shown in the example
4. Choose the placement of the label from the **Placement Bottom Right** drop-down box
5. Make sure the **Enable Copyright Label** checkbox is checked
6. Click **OK**

In the example above, the first line is in bold, the second (created using
) contains a copyright symbol, followed by our company name in italics.

12.2.2 North Arrow Plugin

The North Arrow plugin places a simple north arrow on the map canvas. At present there is only one style available. You can adjust the angle of the arrow or let QGIS set the direction automatically. If you choose to let QGIS determine the direction, it makes its best guess as to how the arrow should be oriented. For placement of the arrow you have four options, corresponding to the four corners of the map canvas.

Figure 38: North Arrow Plugin 




12.2.3 Scale Bar Plugin

The Scale Bar plugin adds a simple scale bar to the map canvas. You control the style and placement, as well as the labeling of the bar.

QGIS only supports displaying the scale in the same units as your map frame. So if the units of your layers are in meters, you can't create a scale bar in feet. Likewise if you are using decimal degrees, you can't create a scale bar to display distance in meters.

To add a scale bar:

1. Click on **Plugins** > **Decorations** > **Scale Bar** or use the  **Scale Bar** button from the Toolbar.


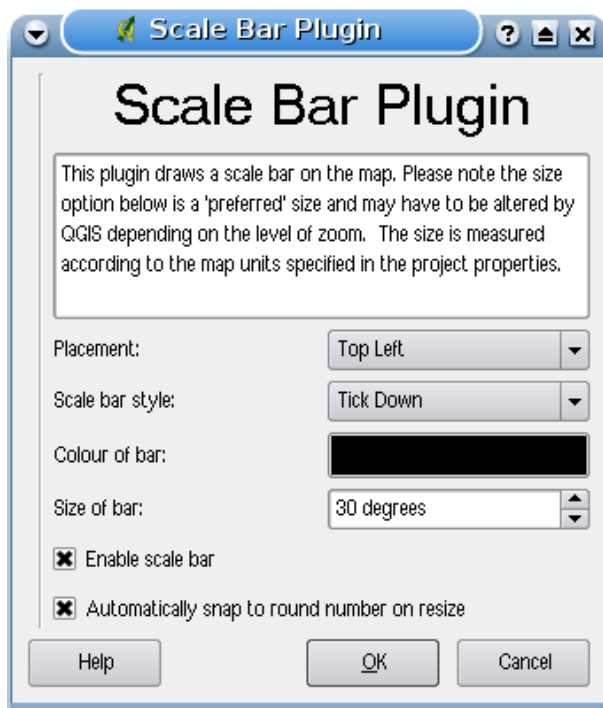
2. Choose the placement from the Placement **Bottom Left** ▼ drop-down list
3. Choose the style from the Scale bar style **Tick Down** ▼ list
4. Select the color for the bar Color of bar  or use the default black color
5. Set the size of the bar and its label Size of bar **30 degrees** ▲▼
6. Make sure the Enable scale bar checkbox is checked
7. Optionally choose to automatically snap to a round number when the canvas is resized Automatically snap to round number on resize
8. Click **OK**

Figure 39: Scale Bar Plugin 



12.3 Delimited Text Plugin

The Delimited Text plugin allows you to load a delimited text file as a layer in QGIS.

Requirements

To view a delimited text file as layer, the text file must contain:

1. A delimited header row of field names. This must be the first line in the text file.
2. The header row must contain an X and Y field. These fields can have any name.
3. The x and y coordinates must be specified as a number. The coordinate system is not important.

As an example of a valid text file we import the elevation point data file `elevp.csv` coming with the QGIS sample dataset (See Section 3.2):

```
X;Y;ELEV
-300120;7689960;13
-654360;7562040;52
1640;7512840;3
[...]
```

Some items of note about the text file are:

1. The example text file uses ; as delimiter. Any character can be used to delimit the fields.
2. The first row is the header row. It contains the fields X, Y and ELEV.
3. No quotes (") are used to delimit text fields.
4. The x coordinates are contained in the X field.
5. The y coordinates are contained in the Y field.

Using the Plugin

To use the plugin you must have QGIS running and use the Plugin Manager to load the plugin:

Start QGIS, then open the Plugin Manager by choosing **Plugins** > **Plugin Manager...** The Plugin Manager displays a list of available plugins. Those that are already loaded have a check mark to the left of their name. Click on the checkbox to the left of the **Add Delimited Text Layer** plugin and click **OK** to load it as described in Section 11.1.


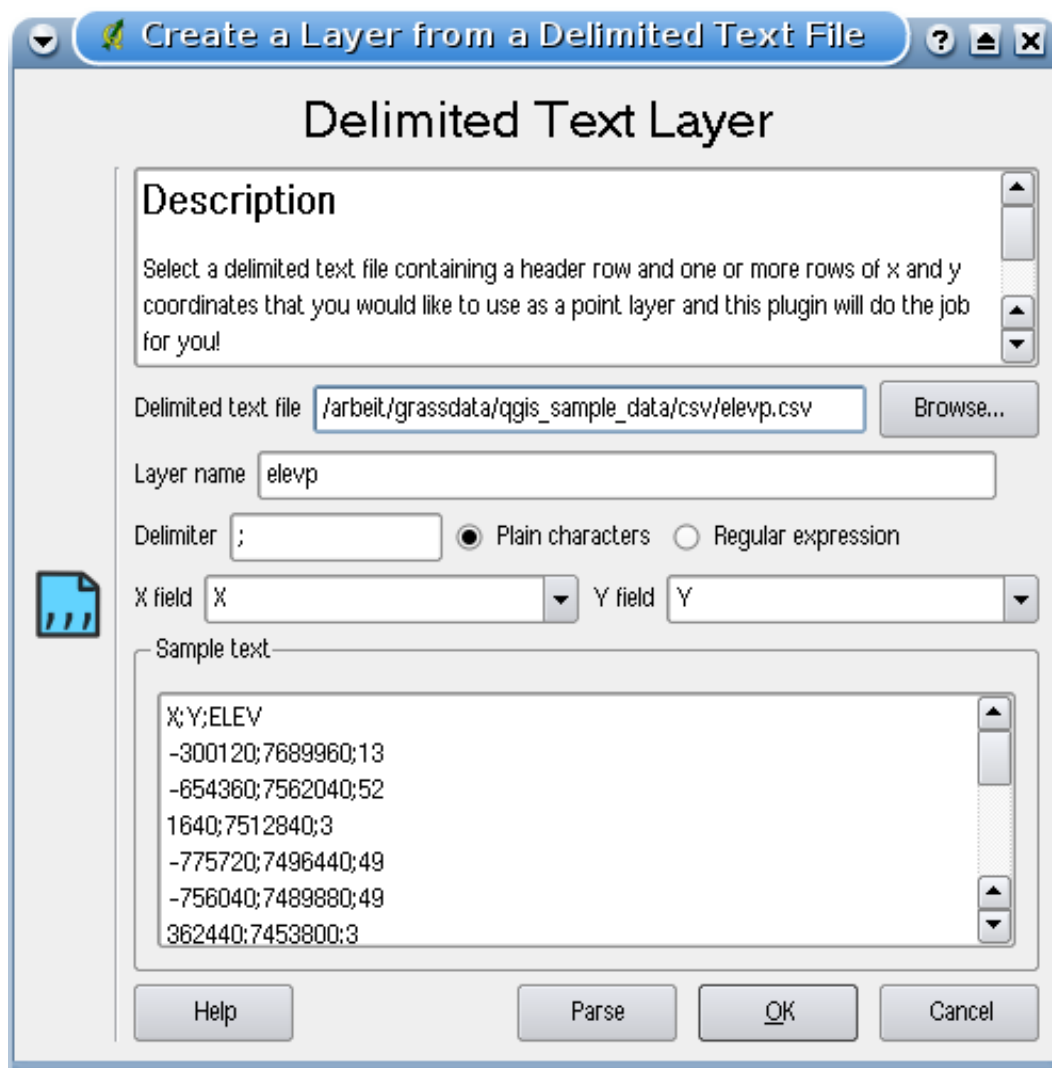
Click the new toolbar icon  **Add Delimited Text Layer** to open the Delimited Text dialog as shown in Figure 40.

Figure 40: Delimited Text Dialog 

First select the file `qgis_sample_data/csv/elevp.csv` to import by clicking on the **Browse** button. Once the file is selected, the plugin attempts to parse the file using the last used delimiter, in this case `;`. To properly parse the file, it is important to select the correct delimiter. To change the delimiter to tab use `\t` (this is a regular expression for the tab character). After changing the delimiter, click **Parse**.

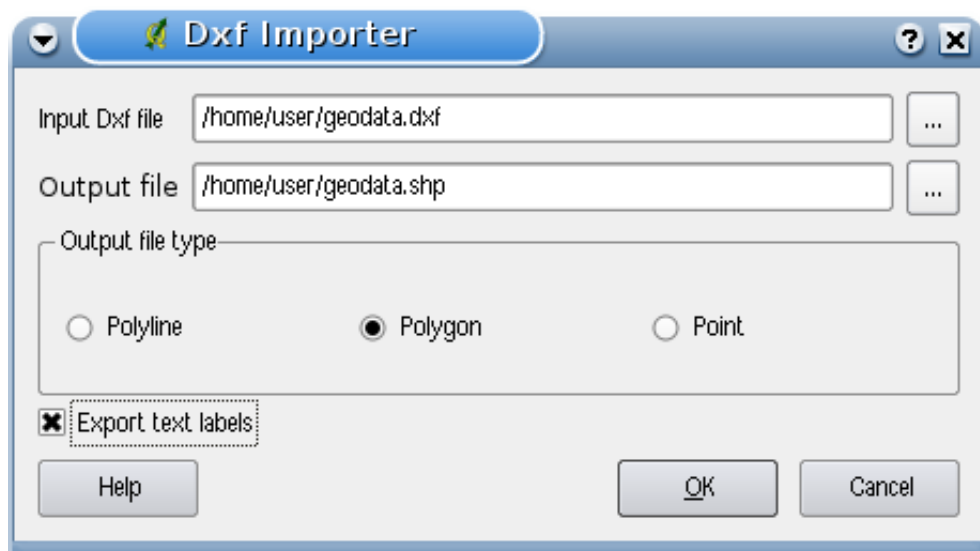
Choose the X and Y fields from the drop down boxes and enter a Layer name `elevp` as shown in Figure 40. To add the layer to the map, click **Add Layer**. The delimited text file now behaves as any other map layer in QGIS.




12.4 Dxf2Shp Converter Plugin

The dxf2shape converter plugin allows to convert vector data from DXF to Shapefile format. It is very simple to handle and provides following functionality as shown in Figure 41.


- **Input DXF file:** Enter path to the DXF file to be converted
- **Output Shp file:** Enter desired name of the shape file to be created
- **Output file type:** specifies the type of the output Shapefile. Currently supported is polyline, polygone and point.
- **Export text labels:** If you enable this checkbox, an additional Shapefile points layer will be created, and the associated dbf table will contain information about the "TEXT" fields found in the dxf file, and the text strings themselves.

Figure 41: Dxf2Shape Converter Plugin 



1. Start QGIS, load the Dxf2Shape plugin in the Plugin Manager (see Section 11.1.1) and click on the  **Dxf2Shape Converter** icon which appears in the QGIS toolbar menu. The Dxf2Shape plugin dialog appears as shown in Figure 41.
2. Enter input DXF file, a name for the output Shapefile and the Shapefile type.
3. Enable the  **Export text labels** checkbox, if you want to create an extra point layer with labels.
4. Click  .

12.5 Georeferencer Plugin

The  **Georeferencer** allows to generate world files for rasters. Therefore you select points on the raster, add their coordinates, and the plugin will compute the world file parameters. The more coordinates you provide the better the result will be.

As an example we will generate a world file for a topo sheet of South Dakota from SDGS. It can later be visualized together with in the data of the GRASS spearfish60 location. You can download the topo sheet here: http://grass.osgeo.org/sampleddata/spearfish_toposheet.tar.gz

As a first step we download the file and untar it.

```
wget http://grass.osgeo.org/sampleddata/spearfish_toposheet.tar.gz
tar xvzf spearfish_toposheet.tar.gz
cd spearfish_toposheet
```

The next step is to start QGIS, load the georeferencer plugin and select the file `spearfish_topo24.tif`.

Figure 42: Select an image to georeference 🐛

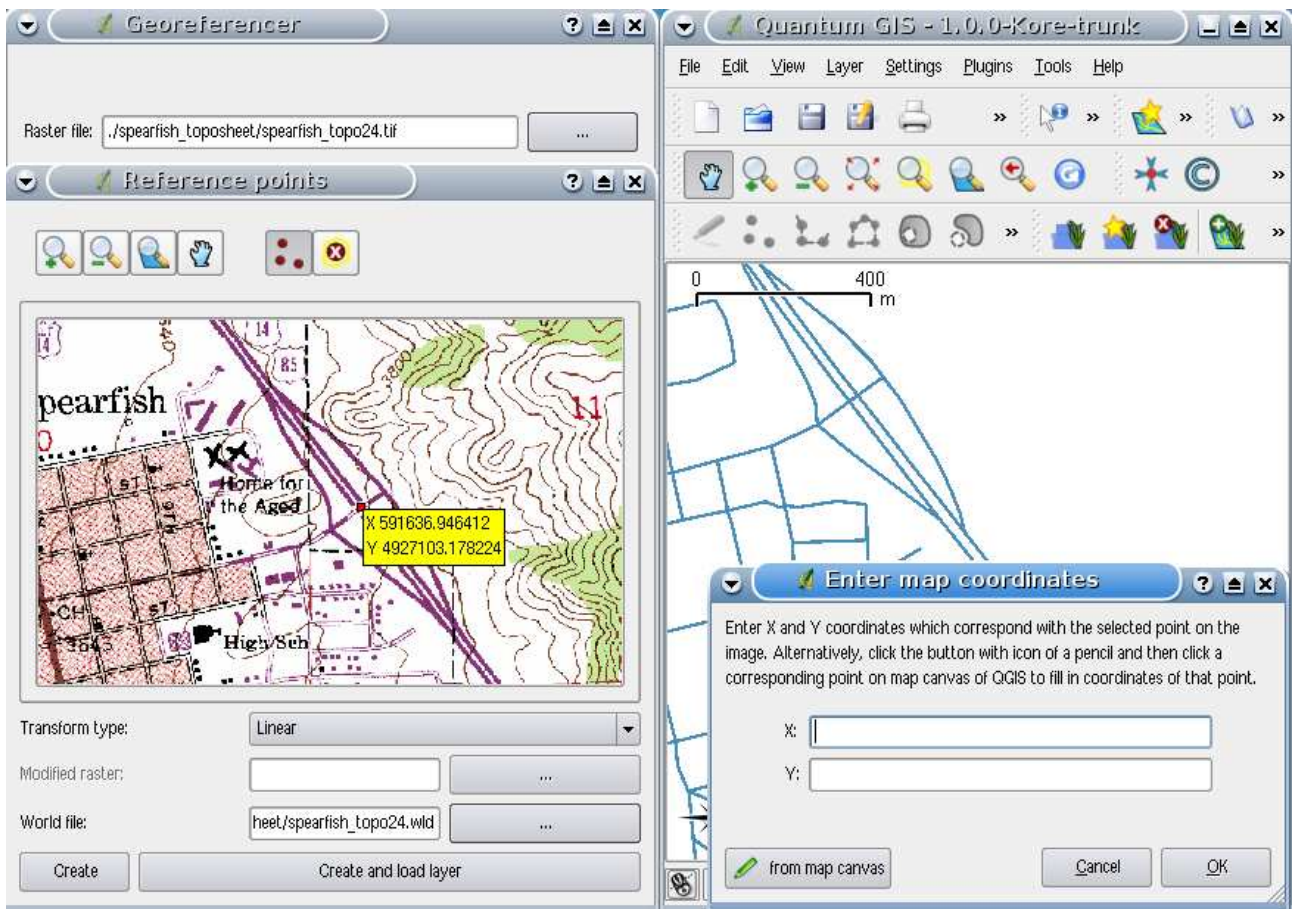


Now click on the button **Arrange plugin window** to open the image in the georeferencer and to arrange it with the reference map in the qgis map canvas on your desktop (see Figure 43).

With the button **Add Point** you can start to add points on the raster image and enter their coordinates, and the plugin will compute the world file parameters (see Figure 44). The more coordinates you provide the better the result will be. For the procedure you have two options:

1. You click on a point in the raster map and enter the X and Y coordinates manually
2. You click on a point in the raster map and choose the button **from map canvas** to add the X and Y coordinates with the help of a georeferenced map already loaded in QGIS.

Figure 43: Arrange plugin window with the qgis map canvas



For this example we use the second option and enter the coordinates for the selected points with the help of the roads map provided with the spearfish60 location from: http://grass.osgeo.org/sampledata/spearfish_grass60data-0.3.tar.gz

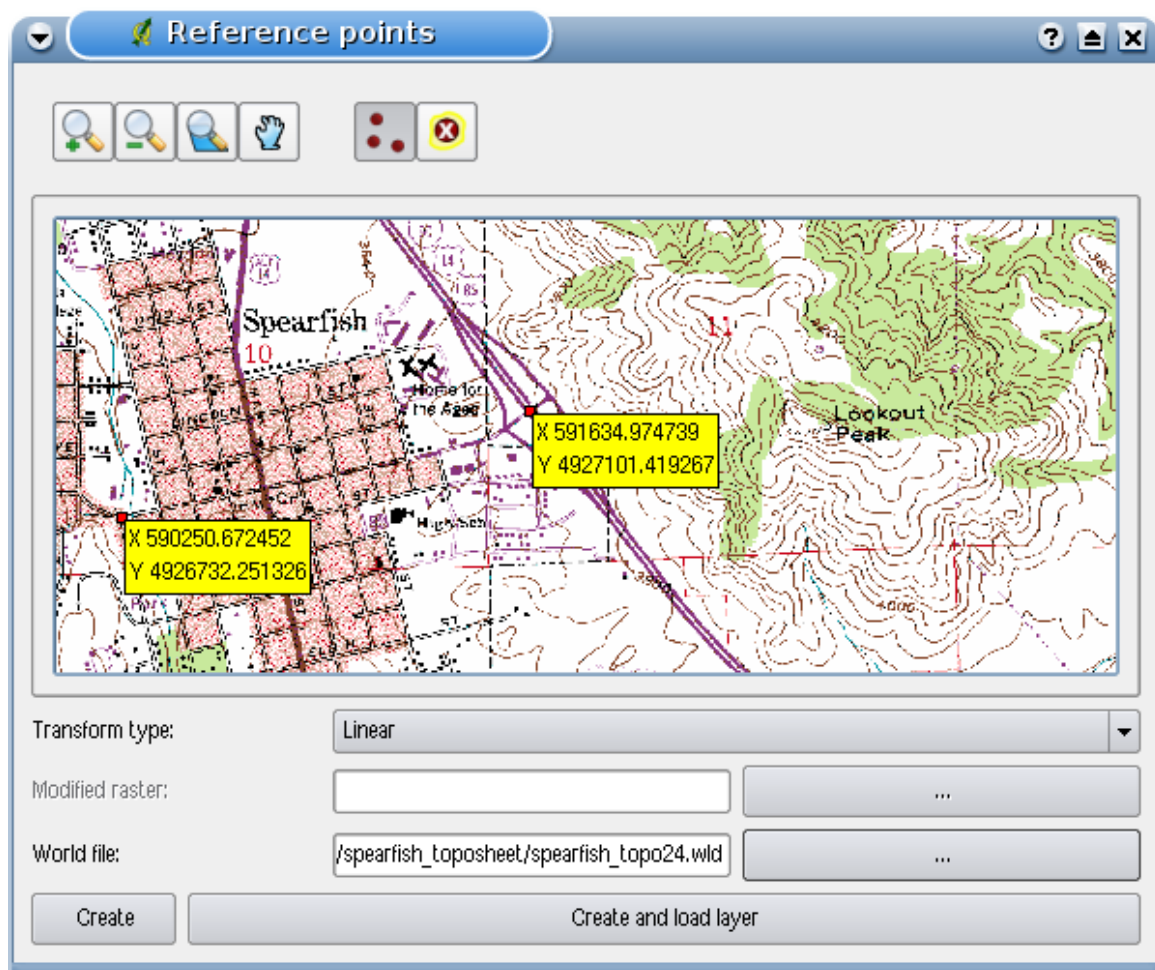
If you don't know how to integrate the spearfish60 location with the GRASS plugin, information are provided in Section 9. As you can see in Figure 44, the georeferencer provides buttons to zoom, pan, add and delete points in the image.

After you added enough points to the image you need to select the transformation type for the georeferencing process and save the resulting world file together with the Tiff. In our example we choose

Transform type although a might be sufficient as well.

The points we added to the map will be stored in a `spearfish_topo24.tif.points` file together with the raster image. This allows us to reopen the georeferencer plugin and to add new points or delete existing ones to optimize the result. The `spearfish_topo24.tif.points` file of this example shows

Figure 44: Add points to the raster image 🐧

**Tip 42** CHOOSING THE TRANSFORMATION TYPE

The linear (affine) transformation is a 1st order transformation and is used for scaling, translation and rotation of geometrically correct images. With the Helmert transformation you simply add coordinate information to the image like geocoding. If your image is contorted you will need to use software that provides 2nd or 3rd order polynomial transformation, e.g. GRASS GIS.

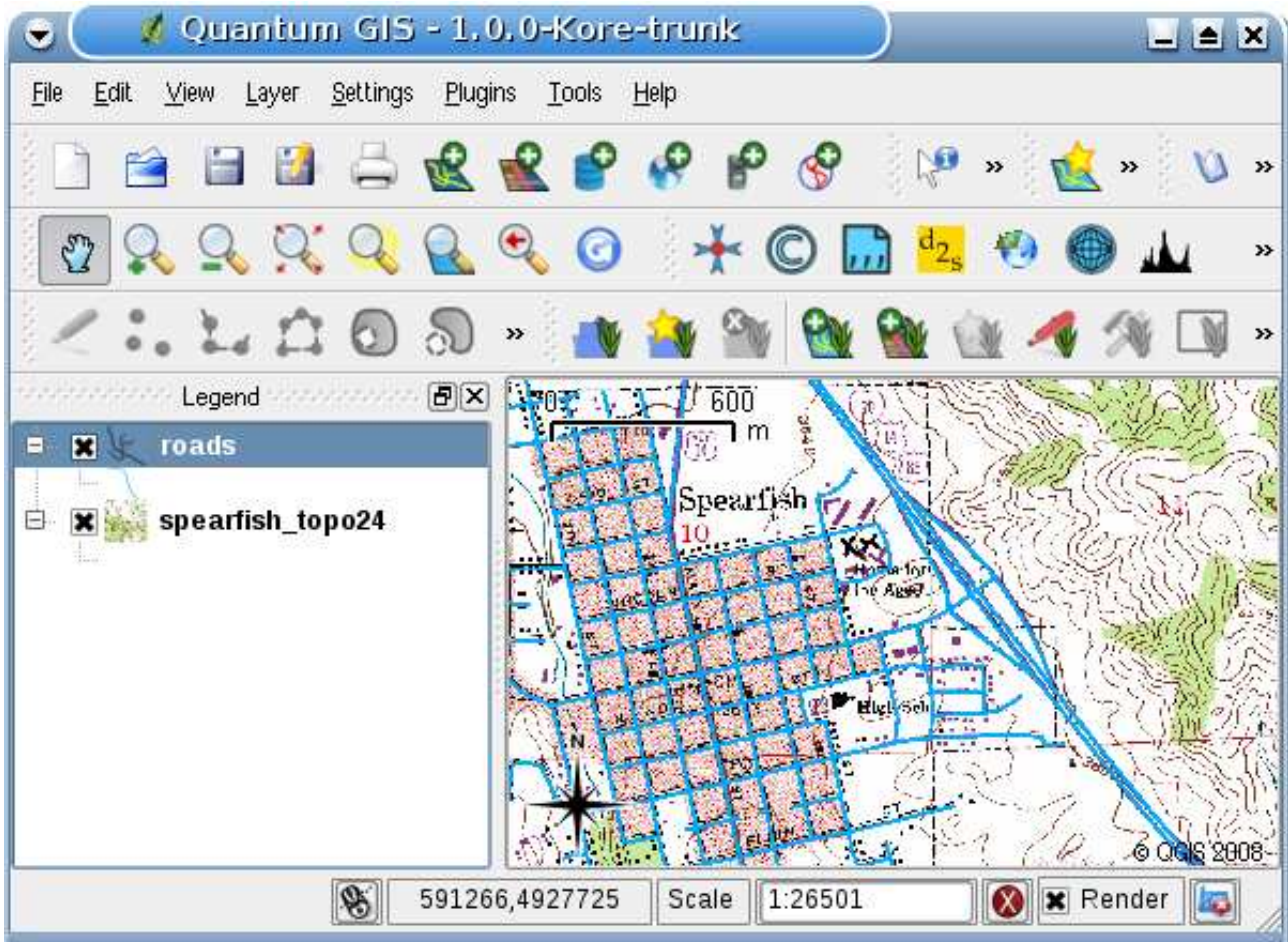
the points:

mapX	mapY	pixelX	pixelY	mapX	mapY
591630.196867999969982	4927104.309682800434530	591647	4.9271e+06		
608453.589164100005291	4924878.995150799863040	608458	4.92487e+06		
602554.903929700027220	4915579.220743400044739	602549	4.91556e+06		
591511.138448899961077	4915952.302661700174212	591563	4.91593e+06		

602649.526155399973504 4919088.353569299913943 602618 4.91907e+06

We used 5 coordinate points to georeference the raster image. To get correct results it is important to disperse the points regularly in the image. Finally we check the result and load the new georeferenced map `spearfish_topo24.tif` and overlay it with the map roads of the `spearfish60` location.

Figure 45: Georeferenced map with overlaid roads from `spearfish60` location 🐱



12.7 GPS Plugin


12.7.1 What is GPS?

GPS, the Global Positioning System, is a satellite-based system that allows anyone with a GPS receiver to find their exact position anywhere in the world. It is used as an aid in navigation, for example in airplanes, in boats and by hikers. The GPS receiver uses the signals from the satellites to calculate its latitude, longitude and (sometimes) elevation. Most receivers also have the capability to store locations (known as *waypoints*), sequences of locations that make up a planned *route* and a tracklog or *track* of the receivers movement over time. Waypoints, routes and tracks are the three basic feature types in GPS data. QGIS displays waypoints in point layers while routes and tracks are displayed in linestring layers.

12.7.2 Loading GPS data from a file

There are dozens of different file formats for storing GPS data. The format that QGIS uses is called GPX (GPS eXchange format), which is a standard interchange format that can contain any number of waypoints, routes and tracks in the same file.

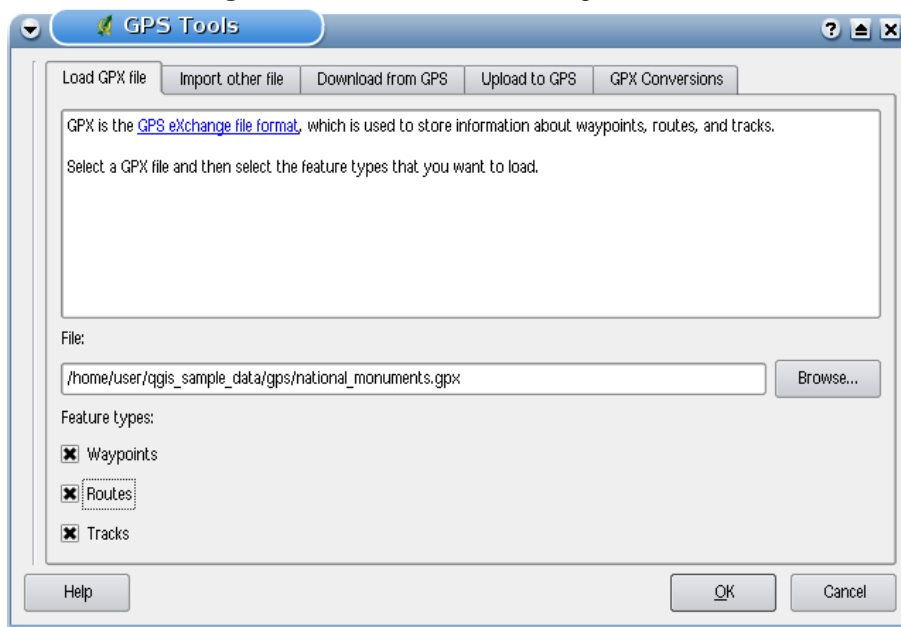
To load a GPX file you first need to load the plugin. **Plugins** > **Plugin Manager...** > **x GPS Tools**. When this plugin is loaded a button with a small handheld GPS device will show up in the toolbar. An example GPX file is available in the QGIS sample dataset: `/qgis_sample_data/gps/national_monuments.gpx`. See Section 3.2 for more information about the sample data.

1. Click on the  **GPS Tools** icon and open the **Load GPX file** tab.
2. **Browse** to the folder `qgis_sample_data/gps/`, select the GPX file `national_monuments.gpx` and click **Open**.

Use the browse button **...** to select the GPX file, then use the checkboxes to select the feature types you want to load from that GPX file. Each feature type will be loaded in a separate layer when you click **OK**. The file `national_monuments.gpx` only includes waypoints.

12.7.3 GPSBabel

Since QGIS uses GPX files you need a way to convert other GPS file formats to GPX. This can be done for many formats using the free program GPSBabel, which is available at <http://www.gpsbabel.org>. This program can also transfer GPS data between your computer and a

Figure 48: The *GPS Tools* dialog window 

GPS device. QGIS uses GPSBabel to do these things, so it is recommended that you install it. However, if you just want to load GPS data from GPX files you will not need it. Version 1.2.3 of GPSBabel is known to work with QGIS, but you should be able to use later versions without any problems.

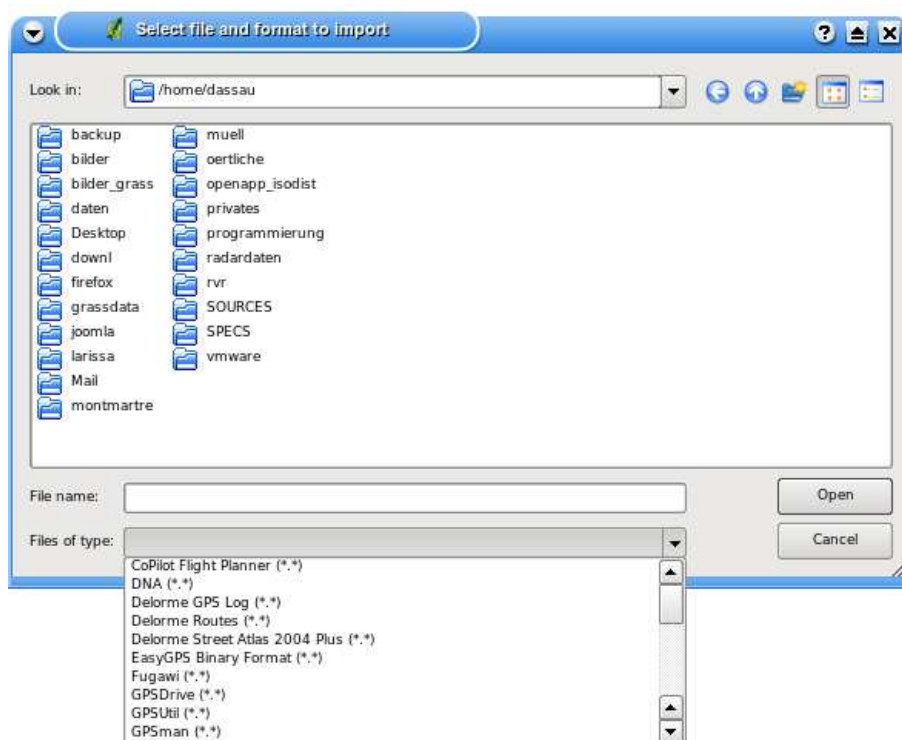

12.7.4 Importing GPS data

To import GPS data from a file that is not a GPX file, you use the tool **Import other file** in the GPS Tools dialog. Here you select the file that you want to import, which feature type you want to import from it, where you want to store the converted GPX file and what the name of the new layer should be.

When you select the file to import you must also select the format of that file by using the menu in the file selection dialog (see figure 49). All formats do not support all three feature types, so for many formats you will only be able to choose between one or two types.



12.7.5 Downloading GPS data from a device

QGIS can use GPSBabel to download data from a GPS device directly into vector layers. For this you use the tool **Download from GPS** (see Figure 50), where you select your type of GPS device, the port that it is connected to, the feature type that you want to download, the GPX file where the

Figure 49: File selection dialog for the import tool 

data should be stored, and the name of the new layer.

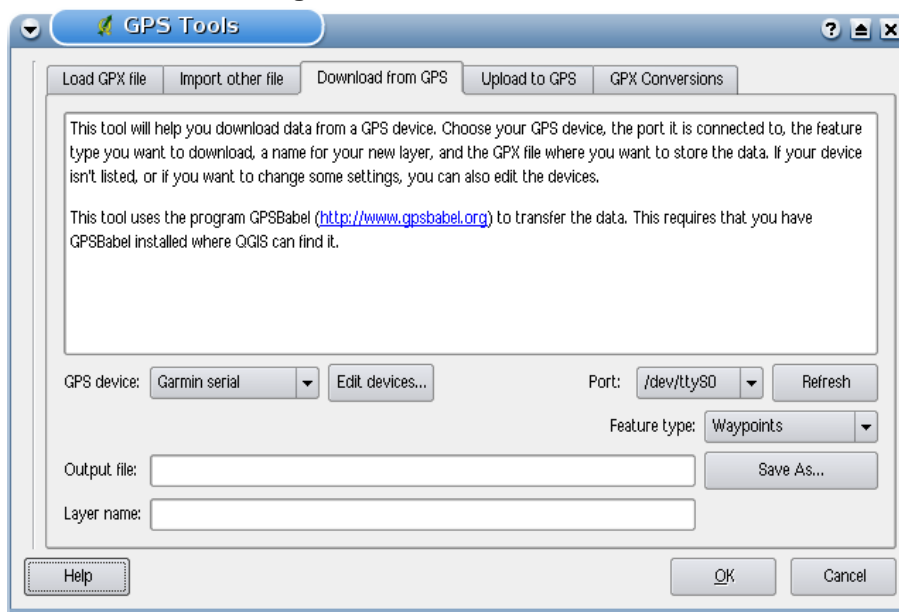

The device type you select in the GPS device menu determines how GPSBabel tries to communicate with the device. If none of the types works with your GPS device you can create a new type (see section 12.7.7).

The port is a file name or some other name that your operating system uses as a reference to the physical port in your computer that the GPS device is connected to.  On Linux this is something like `/dev/ttyS0` or `/dev/ttyS1` and on  Windows it's COM1 or COM2.

When you click **OK** the data will be downloaded from the device and appear as a layer in QGIS.

12.7.6 Uploading GPS data to a device

You can also upload data directly from a vector layer in QGIS to a GPS device, using the tool **Upload to GPS**. The layer must be a GPX layer. To do this you simply select the layer that you want to upload, the type of your GPS device and the port that it's connected to. Just as with the download tool you can specify new device types if your device isn't in the list.

Figure 50: The download tool 


This tool is very useful together with the vector editing capabilities of QGIS. You can load a map, create some waypoints and routes and then upload them and use them in your GPS device.

12.7.7 Defining new device types

There are lots of different types of GPS devices. The QGIS developers can't test all of them, so if you have one that does not work with any of the device types listed in the **Download from GPS** and **Upload to GPS** tools you can define your own device type for it. You do this by using the GPS device editor, which you start by clicking the **Edit devices** button in the download or the upload window.

To define a new device you simply click the **New device** button, enter a name, a download command and an upload command for your device, and click the **Update device** button. The name will be listed in the device menus in the upload and download windows, and can be any string. The download command is the command that is used to download data from the device to a GPX file. This will probably be a GPSSbabel command, but you can use any other command line program that can create a GPX file. QGIS will replace the keywords %type, %in, and %out when it runs the command.

%type will be replaced by “-w” if you are downloading waypoints, “-r” if you are downloading routes and “-t” if you are downloading tracks. These are command line options that tell GPSSbabel which feature type to download.

`%in` will be replaced by the port name that you choose in the download window and `%out` will be replaced by the name you choose for the GPX file that the downloaded data should be stored in. So if you create a device type with the download command `gpsbabel %type -i garmin -o gpx %in %out` (this is actually the download command for the predefined device type GPS device: Garmin serial ) and then use it to download waypoints from port `"/dev/ttyS0"` to the file `"output.gpx"`, QGIS will replace the keywords and run the command `gpsbabel -w -i garmin -o gpx /dev/ttyS0 output.gpx`.


The upload command is the command that is used to upload data to the device. The same keywords are used, but `%in` is now replaced by the name of the GPX file for the layer that is being uploaded, and `%out` is replaced by the port name.

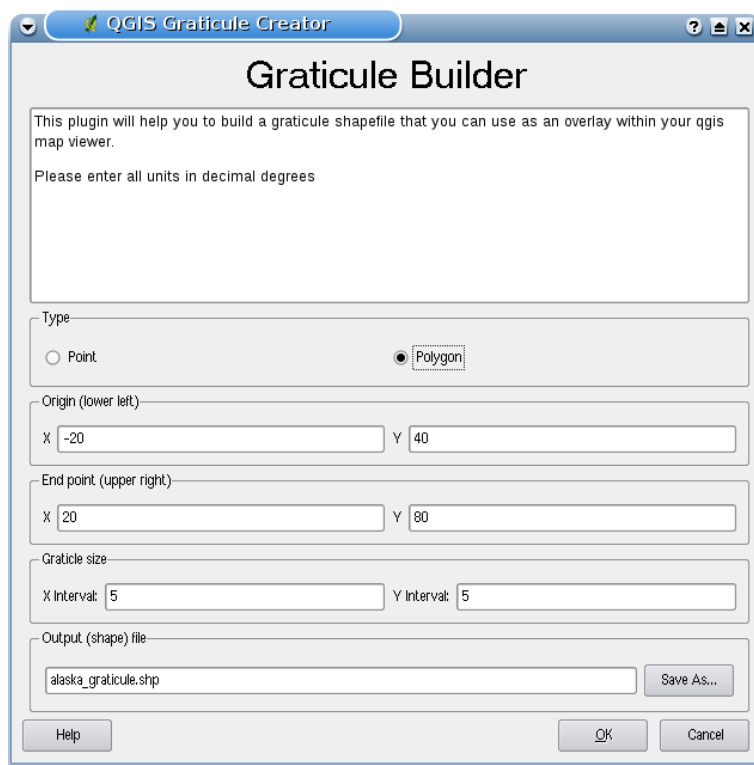
You can learn more about GPSBabel and its available command line options at <http://www.gpsbabel.org>

Once you have created a new device type it will appear in the device lists for the download and upload tools.


12.8 Graticule Creator Plugin

The graticule creator allows to create a “grid” of points or polygons to cover our area of interest. All units must be entered in decimal degrees. The output is a shapefile which can be projected on the fly to match your other data.

Figure 51: Create a graticule layer 



Here is an example how to create a graticule:

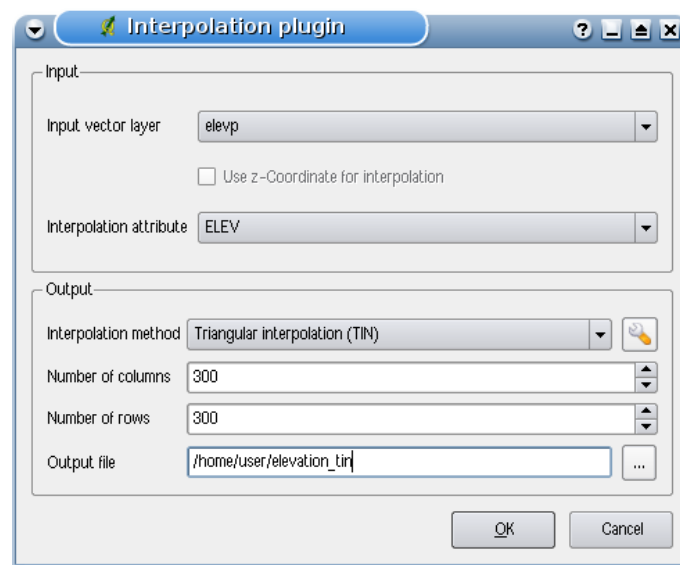
1. Start QGIS, load the Graticule Creator Plugin in the Plugin Manager (see Section 11.1.1) and click on the  Graticule Creator icon which appears in the QGIS toolbar menu.
2. Choose the type of graticule you wish to create: point or polygon.
3. Enter the latitude and longitude for the lower left and upper right corners of the graticule.
4. Enter the interval to be used in constructing the grid. You can enter different values for the X and Y directions (longitude, latitude)
5. Choose the name and location of the shapefile to be created.
6. Click **OK** to create the graticule and add it to the map canvas.


12.9 Interpolation Plugin

The Interpolation plugin allows to interpolate a TIN or IDW raster layer from a vector point layer loaded in the QGIS canvas. It is very simple to handle and provides functionalities as shown in Figure 52.

- **Input vector layer:** Select vector point layer loaded in the QGIS canvas.
- **Interpolation attribute:** Select attribute column used for interpolation or enable Use Z-Coordinate checkbox.
- **Interpolation Method:** Select interpolation method Triangulated Irregular Network (TIN) or Inverse Distance Weighted (IDW).
- **Number of columns/rows:** define number columns and rows for the output raster file
- **Output file:** Define a name for the output raster file

Figure 52: Interpolation Plugin

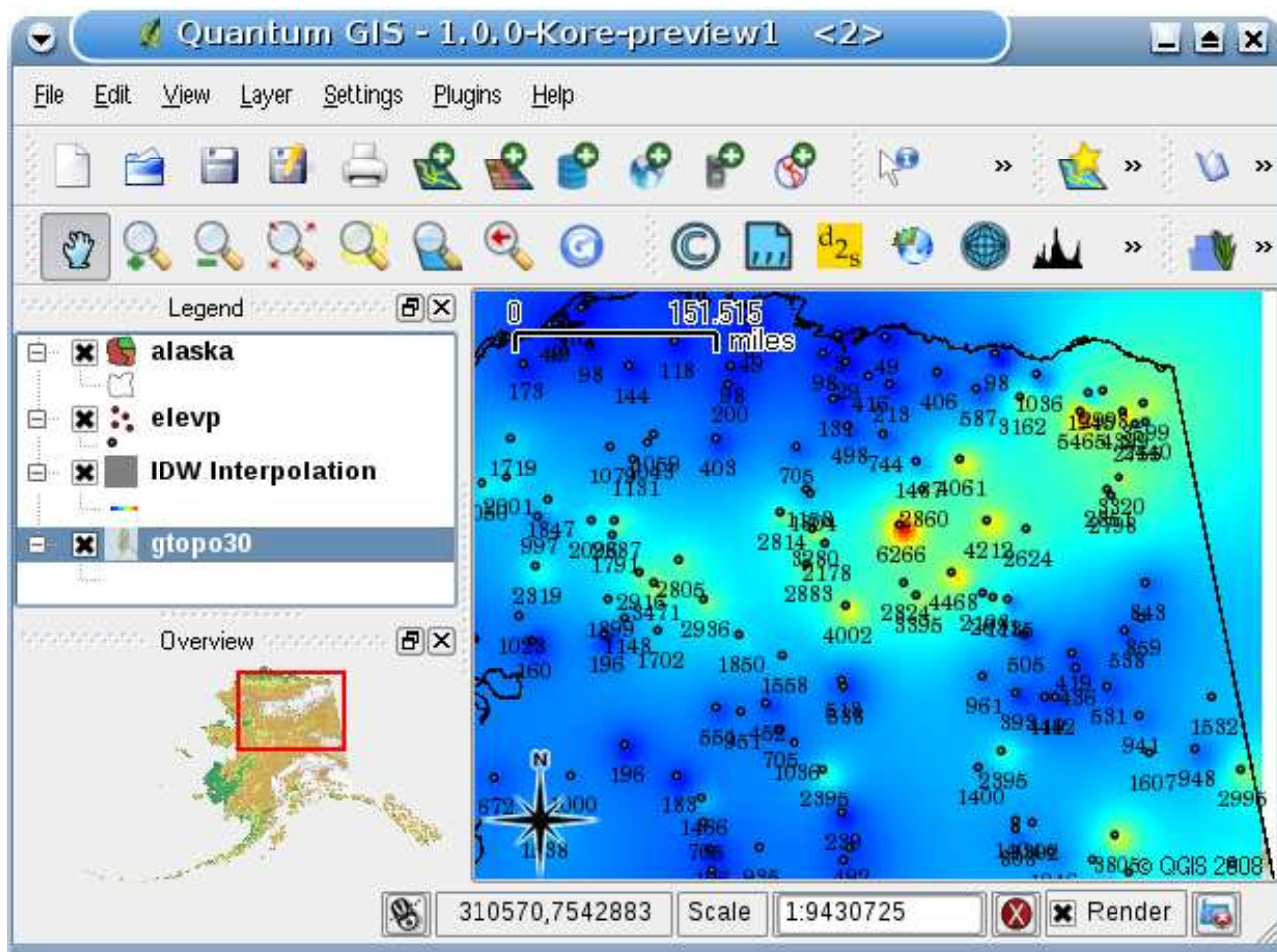


1. Start QGIS and load the `elevp.csv` CSV table with elevation points in the QGIS canvas using the delimited text plugin as described in Section 12.3.
2. Load the Interpolation plugin in the Plugin Manager (see Section 11.1.1) and click on the  **Interpolation** icon which appears in the QGIS toolbar menu. The Interpolation plugin dialog appears as shown in Figure 52.
3. Select `elevp` as input vector and column `ELEV` for interpolation.

4. Select **Triangular interpolation** as interpolation method, define 3663 cols and 1964 rows (this is equivalent to a 1000 meter pixel resolution) as raster output filename `elevation_tin`.
5. Click **Ok**.
6. Double click `elevation_tin` in the map legend to open the Raster Layer Properties dialog and select **Pseudocolor** as Color Map in the **Symbology** tab. Or you can define a new color table as described in Section 6.3.

In Figure 53 you see the IDW interpolation result with a 366 cols x 196 rows (10 km) resolution for the `elevp.csv` data visualized using the Pseudocolor color table. The processing takes a couple of minutes, although the data only cover the northern part of Alaska.

Figure 53: Interpolation of elevp data using IDW method 🗺️



12.10 MapServer Export Plugin

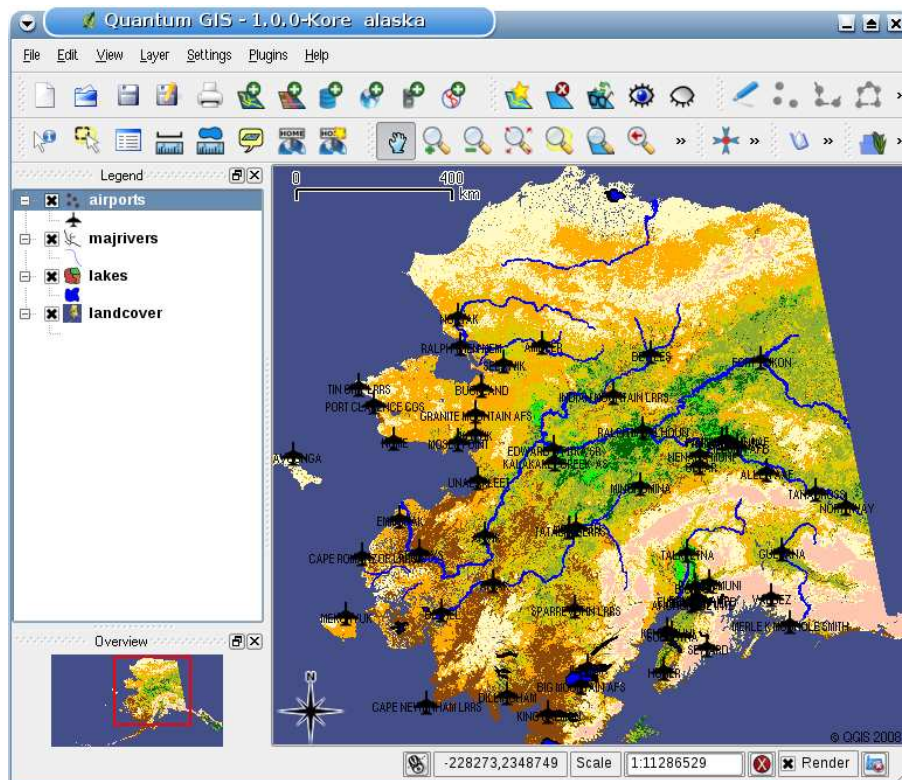
You can use QGIS to “compose” your map by adding and arranging layers, symbolizing them, customizing the colors and then create a map file for MapServer. In order to use the MapServer Export plugin, you must have Python \geq 2.4 installed on your system and QGIS must have been compiled with support for it. All binary packages include Python Support.

The MapServer Export plugin in QGIS 1.0.0 is a Python Plugin, that is automatically loaded into the Plugin Manager as a core plugin (see Section 12).

12.10.1 Creating the Project File



The MapServer Export Plugin operates on a saved QGIS project file and **not** on the current contents of the map canvas and legend. This has been a source of confusion for a number of people. As described below, before you start using the MapServer Export Plugin, you need to arrange the raster and vector layers you want to use in MapServer and save this status in a QGIS project file

Figure 54: Arrange raster and vector layers for QGIS project file 



In this example we show the four steps to get us to the point where we are ready to create the

MapServer map file. We use raster and vector files from the QGIS sample dataset 3.2.

1. Add the raster layer `landcover.tif` clicking on the  **Add Raster Layer** icon.
2. Add the vector Shapefiles `lakes.shp`, `majrivers.shp` and `airports.shp` from the QGIS sample dataset clicking on the  **Add Vector Layer** icon.
3. Change the colors and symbolize the data as you like (see Figure 54)
4. Save a new project named `mapserverproject.qgs` using **File** > **Save Project**.

12.10.2 Creating the Map File

The tool `msexport` to export a QGIS project file to a MapServer map file is installed in your QGIS binary directory and can be used independently of QGIS. From QGIS you need to load the MapServer Export Plugin first with the Plugin Manager. Click **Plugins** > **Manage Plugins...** to open the Plugin


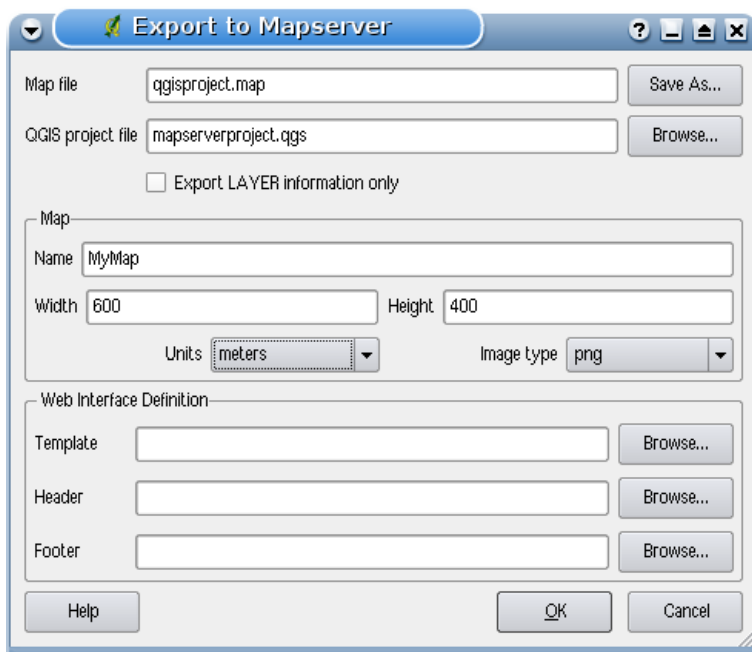
Manager, choose MapServer export Plugin and click **OK**. Now start the  **MapServer Export** dialog (see Figure 55) clicking the icon in the toolbar menu.

Figure 55: Export to MapServer Dialog 



Map file

Enter the name for the map file to be created. You can use the button at the right to browse for

the directory where you want the map file created.

Qgis project file

Enter the full path to the QGIS project file (.qgs) you want to export. You can use the button at the right to browse for the QGIS project file.

Map Name

A name for the map. This name is prefixed to all images generated by the mapserver.

Map Width

Width of the output image in pixels.

Map Height

Height of the output image in pixels.

Map Units

Units of measure used for output

Image type

Format for the output image generated by MapServer

Web Template

Full path to the MapServer template file to be used with the map file


Web Header

Full path to the MapServer header file to be used with the map file

Web Footer

Full path to the MapServer footer file to be used with the map file

Only the `Map file` and `QGIS project file` inputs are required to create a map file, however you may end up with a non-functional map file, depending on your intended use. Although QGIS is good at creating a map file from your project file, it may require some tweaking to get the results you want. But let's create a map file using the project file `mapserverproject.qgs` we just created (see Figure 55):

1. Open the MapServer Export Plugin clicking the  **MapServer Export** icon.
2. Enter the name `qgisproject.map` for your new map file.
3. Browse and find the QGIS project file `mapserverproject.qgs` you just saved.
4. Enter a name `MyMap` for the map.
5. Enter 600 for the width and 400 for the height.
6. Our layers are in meters so we change the units to meters.
7. Choose "png" for the image type.
8. Click **OK** to generate the new map file `qgisproject.map`. QGIS displays the success of your efforts.

You can view the map file in a text editor or visualizer. If you take a look, you'll notice that the export tool adds the metadata needed to enable our map file for WMS.

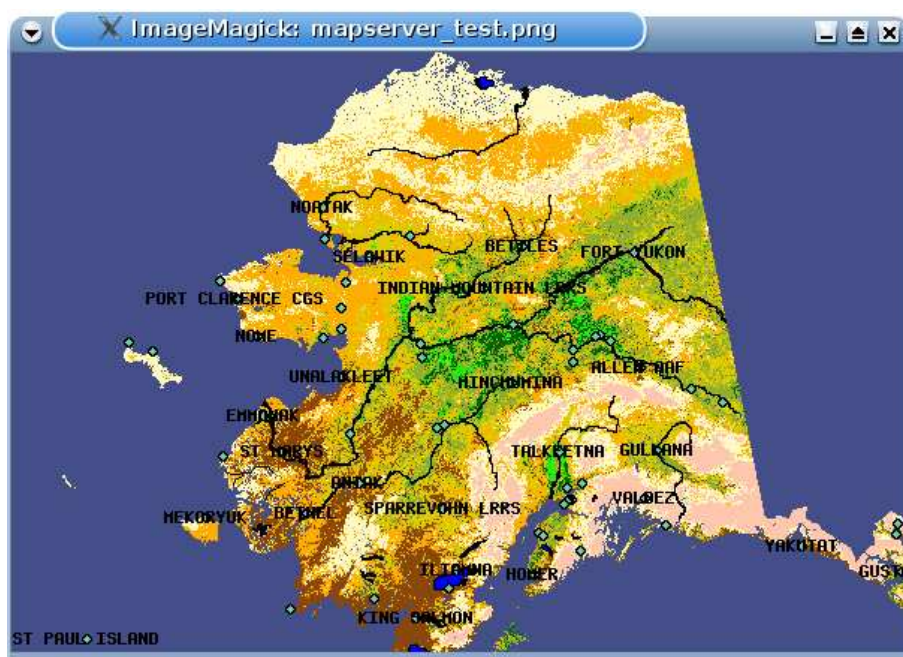
12.10.3 Testing the Map File

We can now test our work using the `shp2img` tool to create an image from the map file. The `shp2img` utility is part of MapServer and FWTools. To create an image from our map:

- Open a terminal window
- If you didn't save your map file in your home directory, change to the folder where you saved it
- Run `shp2img -m qgisproject.map -o mapserver_test.png` and display the image

This creates a PNG with all the layers included in the QGIS project file. In addition, the extent of the PNG will be the same as when we saved the project. As you can see in Figure 56, all information except the airport symbols are included.

Figure 56: Test PNG created by `shp2img` with all MapServer Export layers 



If you plan to use the map file to serve WMS requests, you probably don't have to tweak anything. If you plan to use it with a mapping template or a custom interface, you may have a bit of manual work to do. To see how easy it is to go from QGIS to serving maps on the web, take a look at Christopher Schmidt's 5 minute flash video. He used QGIS version 0.8, but it is still useful. ⁹

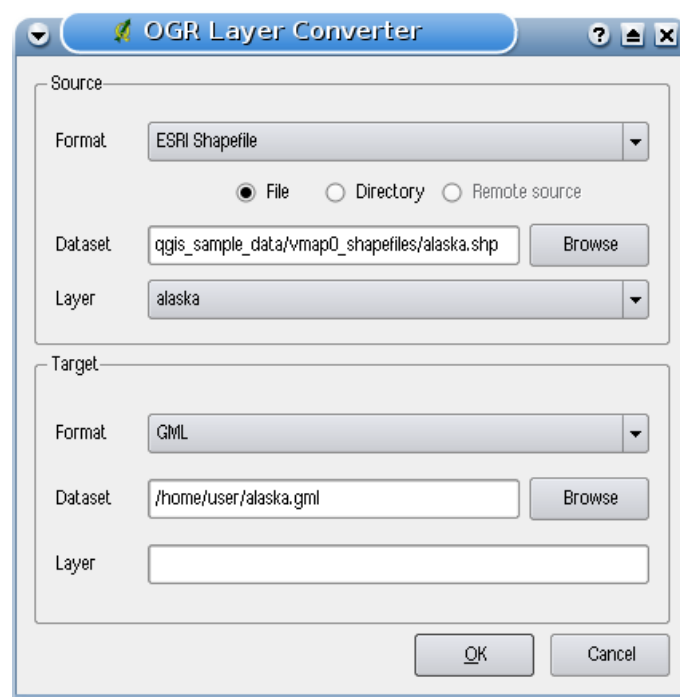
⁹<http://openlayers.org/presentations/mappingyourdata/>




12.11 OGR Converter Plugin

The OGR layer converter plugin allows to convert vector data from one OGR-supported vector format to another OGR-supported vector format. It is very simple to handle and provides functionalities as shown in Figure 57. The supported formats can vary according to the installed GDAL/OGR package.

- **Source Format/Dataset/Layer:** Enter OGR format and path to the vector file to be converted
- **Target Format/Dataset/Layer:** Enter OGR format and path to the vector output file

Figure 57: OGR Layer Converter Plugin 



1. Start QGIS, load the OGR converter plugin in the Plugin Manager (see Section 11.1.1) and click on the  **OGR Layer Converter** icon which appears in the QGIS toolbar menu. The OGR Layer Converter plugin dialog appears as shown in Figure 57.
2. Select the OGR-supported format **ESRI Shapefile**  and the path to the vector input file `alaska.shp` in the Source area.
3. Select the OGR-supported format **GML**  and define a path and the vector output filename `alaska.gml` in the Target area.
4. Click **Ok**.


13 Using external QGIS Python Plugins

External QGIS plugins are written in python. They are stored in an official, moderated repository and maintained by the individual author. Table 7 shows a list of plugins currently available with a short description.^{10 11}

When this manual was released, the external moderated QGIS plugin repository was not fully established. A detailed documentation about the usage, the author and other important information are provided with the external plugin itself and is not part of this manual.

You find an up-to-date list of moderated external plugins in the QGIS Official Repository of the  and at <http://qgis.osgeo.org/download/plugins.html>.


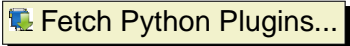




Table 7: Current moderated external QGIS Plugins

Icon	external Plugin	Description
	Zoom To Point	Zooms to a coordinate specified in the input dialog. You can specify the zoom level as well to control the view extent.

A detailed install description for external python plugins can be found in Section 11.1.2.

User-Contributed Python Plugin Repository and author repositories

Apart from the moderated external plugins there exists another unofficial Python Plugin repository. It contains plugins that are not yet mature enough to include them to the official repository, however some of them can be quite useful. Furthermore a few of our contributors maintain their own repositories.

To add the unofficial repository and the author repositories open the Plugin Installer ( > ) go to the  tab and click the  button. If you don't want one or more of the added repositories, disable it with the  button or completely remove with the  button.

Tip 43 ADD MORE EXTERNAL PLUGINS

Apart from the official QGIS plugin repository you can add more external repositories. Therefore select the Repositories tab in the Python Plugins Installer

¹⁰Also updates of core plugins may be available in this repository as external overlays.

¹¹The Python Plugin Installer is also an external Python Plugin, but it is part of the QGIS sources and automatically loaded and selectable inside the QGIS Plugin Manager (see Section 11.1.2).

14 Writing a QGIS Plugin in C++

In this section we provide a beginner's tutorial for writing a simple QGIS C++ plugin. It is based on a workshop held by Dr. Marco Hugentobler.

QGIS C++ plugins are dynamically linked libraries (.so or .dll). They are linked to QGIS at runtime when requested in the plugin manager and extend the functionality of QGIS. They have access to the QGIS GUI and can be divided into core and external plugins.

Technically the QGIS plugin manager looks in the lib/qgis directory for all .so files and loads them when it is started. When it is closed they are unloaded again, except the ones with a checked box. For newly loaded plugins, the *classFactory* method creates an instance of the plugin class and the *initGui* method of the plugin is called to show the GUI elements in the plugin menu and toolbar. The *unload()* function of the plugin is used to remove the allocated GUI elements and the plugin class itself is removed using the class destructor. To list the plugins, each plugin must have a few external 'C' functions for description and of course the *classFactory* method.

14.1 Why C++ and what about licensing

QGIS itself is written in C++, so it also makes sense to write plugins in C++ as well. It is an object-oriented programming (OOP) language that is viewed by many developers as a preferred language for creating large-scale applications.

QGIS C++ plugins use functionalities of libqgis*.so libraries. As they are licensed under GNU GPL, QGIS C++ plugins must be licensed under the GPL, too. This means you may use your plugins for any purpose and you are not forced to publish them. If you do publish them however, they must be published under the conditions of the GPL license.

14.2 Programming a QGIS C++ Plugin in four steps

The example plugin is a point converter plugin and intentionally kept simple. The plugin searches the active vector layer in QGIS, converts all vertices of the layer features to point features keeping the attributes and finally writes the point features into a delimited text file. The new layer can then be loaded into QGIS using the delimited text plugin (see Section [12.3](#)).

Step 1: Make the plugin manager recognise the plugin

As a first step we create the `QgsPointConverter.h` and `QgsPointConverter.cpp` files. Then we add virtual methods inherited from `QgisPlugin` (but leave them empty for now), create necessary external 'C' methods and a `.pro` file, which is a Qt mechanism to easily create Makefiles. Then we compile the sources, move the compiled library into the plugin folder and load it in the QGIS plugin manager.

a) Create new pointconverter.pro file and add:

```
#base directory of the qgis installation
QGIS_DIR = /home/marco/src/qgis

TEMPLATE = lib
CONFIG = qt
QT += xml qt3support
unix:LIBS += -L/$(QGIS_DIR)/lib -lqgis_core -lqgis_gui
INCLUDEPATH += $(QGIS_DIR)/src/ui $(QGIS_DIR)/src/plugins $(QGIS_DIR)/src/gui \
    $(QGIS_DIR)/src/raster $(QGIS_DIR)/src/core $(QGIS_DIR)
SOURCES = qgspointconverterplugin.cpp
HEADERS = qgspointconverterplugin.h
DEST = pointconverterplugin.so
DEFINES += GUI_EXPORT= CORE_EXPORT=
```

b) Create new qgspointconverterplugin.h file and add:

```
#ifndef QGSPOINTCONVERTERPLUGIN_H
#define QGSPOINTCONVERTERPLUGIN_H

#include "qgisplugin.h"

/**A plugin that converts vector layers to delimited text point files.
 The vertices of polygon/line type layers are converted to point features*/
class QgsPointConverterPlugin: public QgsPlugin
{
public:
    QgsPointConverterPlugin(QgisInterface* iface);
    ~QgsPointConverterPlugin();
    void initGui();
    void unload();

private:
    QgisInterface* mIface;
};
#endif
```

c) Create new qgspointconverterplugin.cpp file and add:

```
#include "qgspointconverterplugin.h"
```

```
#ifdef WIN32
#define QGISEXTERN extern "C" __declspec( dllexport )
#else
#define QGISEXTERN extern "C"
#endif

QgsPointConverterPlugin::QgsPointConverterPlugin(QgisInterface* iface): mIface(iface)
{
}

QgsPointConverterPlugin::~QgsPointConverterPlugin()
{
}

void QgsPointConverterPlugin::initGui()
{
}

void QgsPointConverterPlugin::unload()
{
}

QGISEXTERN QgisPlugin* classFactory(QgisInterface* iface)
{
    return new QgsPointConverterPlugin(iface);
}

QGISEXTERN QString name()
{
    return "point converter plugin";
}

QGISEXTERN QString description()
{
    return "A plugin that converts vector layers to delimited text point files";
}

QGISEXTERN QString version()
{
    return "0.00001";
}
```

```
// Return the type (either UI or MapLayer plugin)
QGISEXTERN int type()
{
    return QgisPlugin::UI;
}

// Delete ourself
QGISEXTERN void unload(QgisPlugin* theQgsPointConverterPluginPointer)
{
    delete theQgsPointConverterPluginPointer;
}
```

Step 2: Create an icon, a button and a menu for the plugin

This step includes adding a pointer to the QgisInterface object in the plugin class. Then we create a QAction and a callback function (slot), add it to the QGIS GUI using QgisIface::addToolBarIcon() and QgisIface::addPluginToMenu() and finally remove the QAction in the *unload()* method.

d) Open qgspointconverterplugin.h again and extend existing content to:

```
#ifndef QGSPOINTCONVERTERPLUGIN_H
#define QGSPOINTCONVERTERPLUGIN_H

#include "qgisplugin.h"
#include <QObject>

class QAction;

/**A plugin that converts vector layers to delimited text point files.
 The vertices of polygon/line type layers are converted to point features*/
class QgsPointConverterPlugin: public QObject, public QgisPlugin
{
    Q_OBJECT

public:
    QgsPointConverterPlugin(QgisInterface* iface);
    ~QgsPointConverterPlugin();
    void initGui();
    void unload();

private:
```

```
QgisInterface* mIface;
QAction* mAction;

private slots:
void convertToPoint();
};

#endif
```

e) Open qgspointconverterplugin.cpp again and extend existing content to:

```
#include "qgspointconverterplugin.h"
#include "qgisinterface.h"
#include <QAction>

#ifdef WIN32
#define QGISEXTERN extern "C" __declspec( dllexport )
#else
#define QGISEXTERN extern "C"
#endif

QgsPointConverterPlugin::QgsPointConverterPlugin(QgisInterface* iface): \
    mIface(iface), mAction(0)
{

}

QgsPointConverterPlugin::~QgsPointConverterPlugin()
{

}

void QgsPointConverterPlugin::initGui()
{
    mAction = new QAction(tr("&Convert to point"), this);
    connect(mAction, SIGNAL(activated()), this, SLOT(convertToPoint()));
    mIface->addToolBarIcon(mAction);
    mIface->addPluginToMenu(tr("&Convert to point"), mAction);
}

void QgsPointConverterPlugin::unload()
```

```
{
    mIface->removeToolBarIcon(mAction);
    mIface->removePluginMenu(tr("&Convert to point"), mAction);
    delete mAction;
}

void QgsPointConverterPlugin::convertToPoint()
{
    qWarning("in method convertToPoint");
}

QGISEXTERN QgisPlugin* classFactory(QgisInterface* iface)
{
    return new QgsPointConverterPlugin(iface);
}

QGISEXTERN QString name()
{
    return "point converter plugin";
}

QGISEXTERN QString description()
{
    return "A plugin that converts vector layers to delimited text point files";
}

QGISEXTERN QString version()
{
    return "0.00001";
}

// Return the type (either UI or MapLayer plugin)
QGISEXTERN int type()
{
    return QgisPlugin::UI;
}

// Delete ourself
QGISEXTERN void unload(QgisPlugin* theQgsPointConverterPluginPointer)
{
    delete theQgsPointConverterPluginPointer;
}
```

Step 3: Read point features from the active layer and write to text file

To read the point features from the active layer we need to query the current layer and the location for the new text file. Then we iterate through all features of the current layer, convert the geometries (vertices) to points, open a new file and use QTextStream to write the x- and y-coordinates into it.

f) Open qgsgeometry.h again and extend existing content to

```
class QgsGeometry;
class QTextStream;

private:

void convertPoint(QgsGeometry* geom, const QString& attributeString, \
    QTextStream& stream) const;
void convertMultiPoint(QgsGeometry* geom, const QString& attributeString, \
    QTextStream& stream) const;
void convertLineString(QgsGeometry* geom, const QString& attributeString, \
    QTextStream& stream) const;
void convertMultiLineString(QgsGeometry* geom, const QString& attributeString, \
    QTextStream& stream) const;
void convertPolygon(QgsGeometry* geom, const QString& attributeString, \
    QTextStream& stream) const;
void convertMultiPolygon(QgsGeometry* geom, const QString& attributeString, \
    QTextStream& stream) const;
```

g) Open qgsgeometry.cpp again and extend existing content to:

```
#include "qgsgeometry.h"
#include "qgsvectorprovider.h"
#include "qgsvectorlayer.h"
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>

void QgsPointConverterPlugin::convertToPoint()
{
    qWarning("in method convertToPoint");
    QgsMapLayer* theMapLayer = mIface->activeLayer();
    if(!theMapLayer)
    {
        QMessageBox::information(0, tr("no active layer"), \
```

```
        tr("this plugin needs an active point vector layer to make conversions \
            to points"), QMessageBox::Ok);
    return;
}
QgsVectorLayer* theVectorLayer = dynamic_cast<QgsVectorLayer*>(theMapLayer);
if(!theVectorLayer)
{
    QMessageBox::information(0, tr("no vector layer"), \
        tr("this plugin needs an active point vector layer to make conversions \
            to points"), QMessageBox::Ok);
    return;
}

QString fileName = QFileDialog::getSaveFileName();
if(!fileName.isNull())
{
    qWarning("The selected filename is: " + fileName);
    QFile f(fileName);
    if(!f.open(QIODevice::WriteOnly))
    {
        QMessageBox::information(0, "error", "Could not open file", QMessageBox::Ok);
        return;
    }
    QTextStream theTextStream(&f);
    theTextStream.setRealNumberNotation(QTextStream::FixedNotation);

    QgsFeature currentFeature;
    QgsGeometry* currentGeometry = 0;

    QgsVectorDataProvider* provider = theVectorLayer->dataProvider();
    if(!provider)
    {
        return;
    }

    theVectorLayer->select(provider->attributeIndexes(), \
        theVectorLayer->extent(), true, false);

    //write header
    theTextStream << "x,y";
    theTextStream << endl;
```



```
while(theVectorLayer->nextFeature(currentFeature))
{
QString featureAttributesString;

currentGeometry = currentFeature.geometry();
if(!currentGeometry)
{
continue;
}

switch(currentGeometry->wkbType())
{
case QGis::WKBPoint:
case QGis::WKBPoint25D:
convertPoint(currentGeometry, featureAttributesString, \
theTextStream);
break;

case QGis::WKBMultiPoint:
case QGis::WKBMultiPoint25D:
convertMultiPoint(currentGeometry, featureAttributesString, \
theTextStream);
break;

case QGis::WKBLineString:
case QGis::WKBLineString25D:
convertLineString(currentGeometry, featureAttributesString, \
theTextStream);
break;

case QGis::WKBMultiLineString:
case QGis::WKBMultiLineString25D:
convertMultiLineString(currentGeometry, featureAttributesString \
theTextStream);
break;

case QGis::WKBPolygon:
case QGis::WKBPolygon25D:
convertPolygon(currentGeometry, featureAttributesString, \
theTextStream);
break;
```

```
        case QGis::WKBMultiPolygon:
        case QGis::WKBMultiPolygon25D:
            convertMultiPolygon(currentGeometry, featureAttributesString, \
theTextStream);
            break;
        }
    }
}

//geometry converter functions
void QgsPointConverterPlugin::convertPoint(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsPoint p = geom->asPoint();
    stream << p.x() << "," << p.y();
    stream << endl;
}

void QgsPointConverterPlugin::convertMultiPoint(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsMultiPoint mp = geom->asMultiPoint();
    QgsMultiPoint::const_iterator it = mp.constBegin();
    for(; it != mp.constEnd(); ++it)
    {
        stream << (*it).x() << "," << (*it).y();
        stream << endl;
    }
}

void QgsPointConverterPlugin::convertLineString(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsPolyline line = geom->asPolyline();
    QgsPolyline::const_iterator it = line.constBegin();
    for(; it != line.constEnd(); ++it)
    {
        stream << (*it).x() << "," << (*it).y();
        stream << endl;
    }
}
```

```
void QgsPointConverterPlugin::convertMultiLineString(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsMultiPolyline ml = geom->asMultiPolyline();
    QgsMultiPolyline::const_iterator lineIt = ml.constBegin();
    for(; lineIt != ml.constEnd(); ++lineIt)
    {
        QgsPolyline currentPolyline = *lineIt;
        QgsPolyline::const_iterator vertexIt = currentPolyline.constBegin();
        for(; vertexIt != currentPolyline.constEnd(); ++vertexIt)
        {
            stream << (*vertexIt).x() << "," << (*vertexIt).y();
            stream << endl;
        }
    }
}
```

```
void QgsPointConverterPlugin::convertPolygon(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsPolygon polygon = geom->asPolygon();
    QgsPolygon::const_iterator it = polygon.constBegin();
    for(; it != polygon.constEnd(); ++it)
    {
        QgsPolyline currentRing = *it;
        QgsPolyline::const_iterator vertexIt = currentRing.constBegin();
        for(; vertexIt != currentRing.constEnd(); ++vertexIt)
        {
            stream << (*vertexIt).x() << "," << (*vertexIt).y();
            stream << endl;
        }
    }
}
```

```
void QgsPointConverterPlugin::convertMultiPolygon(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsMultiPolygon mp = geom->asMultiPolygon();
    QgsMultiPolygon::const_iterator polyIt = mp.constBegin();
    for(; polyIt != mp.constEnd(); ++polyIt)
    {
```

```
QgsPolygon currentPolygon = *polyIt;
QgsPolygon::const_iterator ringIt = currentPolygon.constBegin();
for(; ringIt != currentPolygon.constEnd(); ++ringIt)
{
    QgsPolyline currentPolyline = *ringIt;
    QgsPolyline::const_iterator vertexIt = currentPolyline.constBegin();
    for(; vertexIt != currentPolyline.constEnd(); ++vertexIt)
    {
        stream << (*vertexIt).x() << "," << (*vertexIt).y();
        stream << endl;
    }
}
}
```

Step 4: Copy the feature attributes to the text file

At the end we extract the attributes from the active layer using `QgsVectorDataProvider::fieldNameMap()`. For each feature we extract the field values using `QgsFeature::attributeMap()` and add the contents comma separated behind the x- and y-coordinates for each new point feature. For this step there is no need for any further change in `qgspointconverterplugin.h`

h) Open `qgspointconverterplugin.cpp` again and extend existing content to:

```
#include "qgspointconverterplugin.h"
#include "qgisinterface.h"
#include "qgsgeometry.h"
#include "qgsvectordataproducer.h"
#include "qgsvectorlayer.h"
#include <QAction>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>

#ifdef WIN32
#define QGISEXTERN extern "C" __declspec( dllexport )
#else
#define QGISEXTERN extern "C"
#endif

QgsPointConverterPlugin::QgsPointConverterPlugin(QgisInterface* iface): \
mIface(iface), mAction(0)
```

```
{  
  
}  
  
QgsPointConverterPlugin::~QgsPointConverterPlugin()  
{  
  
}  
  
void QgsPointConverterPlugin::initGui()  
{  
    mAction = new QAction(tr("&Convert to point"), this);  
    connect(mAction, SIGNAL(activated()), this, SLOT(convertToPoint()));  
    mIface->addToolBarIcon(mAction);  
    mIface->addPluginToMenu(tr("&Convert to point"), mAction);  
}  
  
void QgsPointConverterPlugin::unload()  
{  
    mIface->removeToolBarIcon(mAction);  
    mIface->removePluginMenu(tr("&Convert to point"), mAction);  
    delete mAction;  
}  
  
void QgsPointConverterPlugin::convertToPoint()  
{  
    qWarning("in method convertToPoint");  
    QgsMapLayer* theMapLayer = mIface->activeLayer();  
    if(!theMapLayer)  
    {  
        QMessageBox::information(0, tr("no active layer"), \  
            tr("this plugin needs an active point vector layer to make conversions \  
                to points"), QMessageBox::Ok);  
        return;  
    }  
    QgsVectorLayer* theVectorLayer = dynamic_cast<QgsVectorLayer*>(theMapLayer);  
    if(!theVectorLayer)  
    {  
        QMessageBox::information(0, tr("no vector layer"), \  
            tr("this plugin needs an active point vector layer to make conversions \  
                to points"), QMessageBox::Ok);  
        return;  
    }  
}
```

```
    }

    QString fileName = QFileDialog::getSaveFileName();
    if(!fileName.isNull())
    {
        qWarning("The selected filename is: " + fileName);
        QFile f(fileName);
        if(!f.open(QIODevice::WriteOnly))
        {
            QMessageBox::information(0, "error", "Could not open file", QMessageBox::Ok);
            return;
        }
        QTextStream theTextStream(&f);
        theTextStream.setRealNumberNotation(QTextStream::FixedNotation);

        QgsFeature currentFeature;
        QgsGeometry* currentGeometry = 0;

        QgsVectorDataProvider* provider = theVectorLayer->dataProvider();
        if(!provider)
        {
            return;
        }

        theVectorLayer->select(provider->attributeIndexes(), \
            theVectorLayer->extent(), true, false);

        //write header
        theTextStream << "x,y";
        QMap<QString, int> fieldMap = provider->fieldNameMap();
        //We need the attributes sorted by index.
        //Therefore we insert them in a second map where key / values are exchanged
        QMap<int, QString> sortedFieldMap;
        QMap<QString, int>::const_iterator fieldIt = fieldMap.constBegin();
        for(; fieldIt != fieldMap.constEnd(); ++fieldIt)
        {
            sortedFieldMap.insert(fieldIt.value(), fieldIt.key());
        }

        QMap<int, QString>::const_iterator sortedFieldIt = sortedFieldMap.constBegin();
        for(; sortedFieldIt != sortedFieldMap.constEnd(); ++sortedFieldIt)
        {
```

```
        theTextStream << "," << sortedFieldIt.value();
    }

    theTextStream << endl;

    while(theVectorLayer->nextFeature(currentFeature))
    {
        QString featureAttributesString;
        const QgsAttributeMap& map = currentFeature.attributeMap();
        QgsAttributeMap::const_iterator attributeIt = map.constBegin();
        for(; attributeIt != map.constEnd(); ++attributeIt)
        {
            featureAttributesString.append(",");
            featureAttributesString.append(attributeIt.value().toString());
        }

        currentGeometry = currentFeature.geometry();
        if(!currentGeometry)
        {
            continue;
        }

        switch(currentGeometry->wkbType())
        {
            case Qgs::WKBPoint:
            case Qgs::WKBPoint25D:
                convertPoint(currentGeometry, featureAttributesString, \
theTextStream);
                break;

            case Qgs::WKBMultiPoint:
            case Qgs::WKBMultiPoint25D:
                convertMultiPoint(currentGeometry, featureAttributesString, \
theTextStream);
                break;

            case Qgs::WKBLineString:
            case Qgs::WKBLineString25D:
                convertLineString(currentGeometry, featureAttributesString, \
theTextStream);
                break;
```

```
        case QGis::WKBMultiLineString:
        case QGis::WKBMultiLineString25D:
            convertMultiLineString(currentGeometry, featureAttributesString \
theTextStream);
            break;

        case QGis::WKBPolygon:
        case QGis::WKBPolygon25D:
            convertPolygon(currentGeometry, featureAttributesString, \
theTextStream);
            break;

        case QGis::WKBMultiPolygon:
        case QGis::WKBMultiPolygon25D:
            convertMultiPolygon(currentGeometry, featureAttributesString, \
theTextStream);
            break;
    }
}

//geometry converter functions
void QgsPointConverterPlugin::convertPoint(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsPoint p = geom->asPoint();
    stream << p.x() << "," << p.y();
    stream << attributeString;
    stream << endl;
}

void QgsPointConverterPlugin::convertMultiPoint(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsMultiPoint mp = geom->asMultiPoint();
    QgsMultiPoint::const_iterator it = mp.constBegin();
    for(; it != mp.constEnd(); ++it)
    {
        stream << (*it).x() << "," << (*it).y();
        stream << attributeString;
    }
}
```



```
        stream << endl;
    }
}

void QgsPointConverterPlugin::convertLineString(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsPolyline line = geom->asPolyline();
    QgsPolyline::const_iterator it = line.constBegin();
    for(; it != line.constEnd(); ++it)
    {
        stream << (*it).x() << "," << (*it).y();
        stream << attributeString;
        stream << endl;
    }
}

void QgsPointConverterPlugin::convertMultiLineString(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsMultiPolyline ml = geom->asMultiPolyline();
    QgsMultiPolyline::const_iterator lineIt = ml.constBegin();
    for(; lineIt != ml.constEnd(); ++lineIt)
    {
        QgsPolyline currentPolyline = *lineIt;
        QgsPolyline::const_iterator vertexIt = currentPolyline.constBegin();
        for(; vertexIt != currentPolyline.constEnd(); ++vertexIt)
        {
            stream << (*vertexIt).x() << "," << (*vertexIt).y();
            stream << attributeString;
            stream << endl;
        }
    }
}

void QgsPointConverterPlugin::convertPolygon(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsPolygon polygon = geom->asPolygon();
    QgsPolygon::const_iterator it = polygon.constBegin();
    for(; it != polygon.constEnd(); ++it)
    {
```

```
    QgsPolyline currentRing = *it;
    QgsPolyline::const_iterator vertexIt = currentRing.constBegin();
    for(; vertexIt != currentRing.constEnd(); ++vertexIt)
    {
        stream << (*vertexIt).x() << "," << (*vertexIt).y();
        stream << attributeString;
        stream << endl;
    }
}

void QgsPointConverterPlugin::convertMultiPolygon(QgsGeometry* geom, const QString& \
attributeString, QTextStream& stream) const
{
    QgsMultiPolygon mp = geom->asMultiPolygon();
    QgsMultiPolygon::const_iterator polyIt = mp.constBegin();
    for(; polyIt != mp.constEnd(); ++polyIt)
    {
        QgsPolygon currentPolygon = *polyIt;
        QgsPolygon::const_iterator ringIt = currentPolygon.constBegin();
        for(; ringIt != currentPolygon.constEnd(); ++ringIt)
        {
            QgsPolyline currentPolyline = *ringIt;
            QgsPolyline::const_iterator vertexIt = currentPolyline.constBegin();
            for(; vertexIt != currentPolyline.constEnd(); ++vertexIt)
            {
                stream << (*vertexIt).x() << "," << (*vertexIt).y();
                stream << attributeString;
                stream << endl;
            }
        }
    }
}

QGISEXTERN QgisPlugin* classFactory(QgisInterface* iface)
{
    return new QgsPointConverterPlugin(iface);
}

QGISEXTERN QString name()
{
    return "point converter plugin";
}
```

```
}

QGISEXTERN QString description()
{
    return "A plugin that converts vector layers to delimited text point files";
}

QGISEXTERN QString version()
{
    return "0.00001";
}

// Return the type (either UI or MapLayer plugin)
QGISEXTERN int type()
{
    return QgisPlugin::UI;
}

// Delete ourself
QGISEXTERN void unload(QgisPlugin* theQgsPointConverterPluginPointer)
{
    delete theQgsPointConverterPluginPointer;
}
```

14.3 Further information

As you can see, you need information from different sources to write QGIS C++ plugins. Plugin writers need to know C++, the QGIS plugin interface as well as Qt4 classes and tools. At the beginning it is best to learn from examples and copy the mechanism of existing plugins.

There is a a collection of online documentation that may be usefull for QGIS C++ programers:

- QGIS Plugin Debugging: <http://wiki.qgis.org/qgiswiki/DebuggingPlugins>
- QGIS API Documentation: http://svn.qgis.org/api_doc/html/
- Qt documentation: <http://doc.trolltech.com/4.3/index.html>

15 Writing a QGIS Plugin in Python

In this section you find a beginner's tutorial for writing a QGIS Python plugins. It is based on the workshop "Extending the Functionality of QGIS with Python Plugins" held at FOSS4G 2008 by Dr. Marco Hugentobler, Dr. Horst Düster and Tim Sutton.

Apart from writing a QGIS Python plugin, it is also possible to use PyQGIS from a python command line console which is mainly interesting for debugging or to write standalone applications in Python with their own user interfaces using the functionality of the QGIS core library.

15.1 Why Python and what about licensing

Python is a scripting language which was designed with the goal of being easy to program. It has a mechanism that automatically releases memory that is no longer used (garbage collector). A further advantage is that many programs that are written in C++ or Java offer the possibility to write extensions in Python, e.g. OpenOffice or Gimp. Therefore it is a good investment of time to learn the Python language.

PyQGIS plugins use functionality of `libqgis_core.so` and `libqgis_gui.so`. As both are licensed under GNU GPL, QGIS Python plugins must be licenced under the GPL, too. This means you may use your plugins for any purpose and you are not forced to publish them. If you do publish them however, they must be published under the conditions of the GPL license.

15.2 What needs to be installed to get started

On the lab computers, everything for the workshop is already installed. If you program Python plugins at home, you will need the following libraries and programs:

- QGIS
- Python
- Qt
- PyQT
- PyQt development tools

If you use Linux, there are binary packages for all major distributions. For Windows, the PyQt installer already contains Qt, PyQt and the PyQt development tools.




15.3 Programming a simple PyQGIS Plugin in four steps

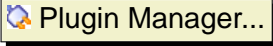
The example plugin is intentionally kept simple. It adds a button to the menu bar of QGIS. If the button is clicked, a file dialog appears where the user may load a shape file.

For each python plugin, a dedicated folder that contains the plugin files needs to be created. By default, QGIS looks for plugins in two locations: `$QGIS_DIR/share/qgis/python/plugins` and `$HOME/.qgis/python/plugins`. Note that plugins installed in the latter location are only visible for one user.

Step 1: Make the plugin manager recognise the plugin

Each Python plugin is contained in its own directory. When QGIS starts up it will scan each OS specific subdirectory and initialize any plugins it finds.

-  Linux and other unices:
`./share/qgis/python/plugins`
`/home/$USERNAME/.qgis/python/plugins`
-  Mac OS X:
`./Contents/MacOS/share/qgis/python/plugins`
`/Users/$USERNAME/.qgis/python/plugins`
-  Windows:
`C:\Program Files\QGIS\python\plugins`
`C:\Documents and Settings\$USERNAME\.qgis\python\plugins`

Once that's done, the plugin will show up in the 

Tip 44 TWO QGIS PYTHON PLUGIN FOLDERS

There are two directories containing the python plugins. `$QGIS_DIR/share/qgis/python/plugins` is designed mainly for the core plugins while `$HOME/.qgis/python/plugins` for easy installation of the external plugins. Plugins in the home location are only visible for one user but also mask the core plugins with the same name, what can be used to provide main plugin updates

To provide the necessary information for QGIS, the plugin needs to implement the methods `name()`, `description()`, `version()`, `qgisMinimumVersion()` and `authorName()` which return descriptive strings. The `qgisMinimumVersion()` should return a simple form, for example "1.0". A plugin also needs a method `classFactory(QgisInterface)` which is called by the plugin manager to create an instance of the plugin. The argument of type `QgisInterface` is used by the plugin to access functions of the QGIS instance. We are going to work with this object in step 2.

Note that, in contrast to other programming languages, indentation is very important. The Python interpreter throws an error if it is not correct.

For our plugin we create the plugin folder 'foss4g_plugin' in \$HOME/.qgis/python/plugins. Then we add two new textfiles into this folder, `foss4gplugin.py` and `__init__.py`.

The file `foss4gplugin.py` contains the plugin class:

```
# -*- coding: utf-8 -*-
# Import the PyQt and QGIS libraries
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
# Initialize Qt resources from file resources.py
import resources

class FOSS4GPlugin:

    def __init__(self, iface):
        # Save reference to the QGIS interface
        self.iface = iface


    def initGui(self):
        print 'Initialising GUI'

    def unload(self):
        print 'Unloading plugin'
```

The file `__init__.py` contains the methods `name()`, `description()`, `version()`, `qgisMinimumVersion()` and `authorName()` and `classFactory`. As we are creating a new instance of the plugin class, we need to import the code of this class:

```
# -*- coding: utf-8 -*-
from foss4gplugin import FOSS4GPlugin
def name():
    return "FOSS4G example"
def description():
    return "A simple example plugin to load shapefiles"
def version():
    return "0.1"
def qgisMinimumVersion():
    return "1.0"
def authorName():
    return "John Developer"
def classFactory(iface):
```

```
return FOSS4GPlugin(iface)
```

At this point the plugin already has the necessary infrastructure to appear in the QGIS  Plugin Manager... to be loaded or unloaded.

Step 2: Create an Icon for the plugin

To make the icon graphic available for our program, we need a so-called resource file. In the resource file, the graphic is contained in hexadecimal notation. Fortunately, we don't need to care about its representation because we use the `pyrcc` compiler, a tool that reads the file `resources.qrc` and creates a resource file.

The file `foss4g.png` and the `resources.qrc` we use in this little workshop can be downloaded from http://karlinapp.ethz.ch/python_foss4g. Move these 2 files into the directory of the example plugin `$HOME/.qgis/python/plugins/foss4g_plugin` and enter there: `pyrcc4 -o resources.py resources.qrc`.

Step 3: Add a button and a menu

In this section, we implement the content of the methods `initGui()` and `unload()`. We need an instance of the class **QAction** that executes the `run()` method of the plugin. With the action object, we are then able to generate the menu entry and the button:

```
import resources

def initGui(self):
    # Create action that will start plugin configuration
    self.action = QAction(QIcon(":/plugins/foss4g_plugin/foss4g.png"), "FOSS4G plugin",
self.iface.getMainWindow())
    # connect the action to the run method
    QObject.connect(self.action, SIGNAL("activated()"), self.run)

    # Add toolbar button and menu item
    self.iface.addToolBarIcon(self.action)
    self.iface.addPluginMenu("FOSS-GIS plugin...", self.action)

def unload(self):
    # Remove the plugin menu item and icon
    self.iface.removePluginMenu("FOSSGIS Plugin...", self.action)
    self.iface.removeToolBarIcon(self.action)
```

Step 4: Load a layer from a shape file

In this step we implement the real functionality of the plugin in the `run()` method. The Qt4 method `QFileDialog::getOpenFileName` opens a file dialog and returns the path to the chosen file. If the user cancels the dialog, the path is a null object, which we test for. We then call the method `addVectorLayer` of the interface object which loads the layer. The method only needs three arguments: the file path, the name of the layer that will be shown in the legend and the data provider name. For shapefiles, this is 'ogr' because QGIS internally uses the OGR library to access shapefiles:

```
def run(self):
    fileName = QFileDialog.getOpenFileName(None,QString.fromLocal8Bit("Select a file:"),
    "", "*.shp *.gml")
    if fileName.isNull():
        QMessageBox.information(None, "Cancel", "File selection canceled")
    else:
        vlayer = self.iface.addVectorLayer(fileName, "myLayer", "ogr")
```

15.4 Committing the plugin to repository

If you have written a plugin you consider to be useful and you want to share with other users you're welcome to upload it to the QGIS User-Contributed Repository.

- Prepare a plugin directory containing only necessary files (ensure that there is no compiled .pyc files, Subversion .svn directories etc).
- Make a zip archive of it, including the directory. Be sure the zip file name is exactly the same as the directory inside (except the .zip extension of course). In other case the Plugin Installer won't be able to relate the available plugin with its locally installed instance.
- Upload it to the repository: <http://pyqgis.org/admin/contributed> (you will need to register at first time). Please pay attention when filling the form. Especially the Version Number field is often filled wrongly what confuses the Plugin Installer and causes false notifications of available updates.

15.5 Further information

As you can see, you need information from different sources to write PyQGIS plugins. Plugin writers need to know Python and the QGIS plugin interface as well as the Qt4 classes and tools. At the beginning it is best to learn from examples and copy the mechanism of existing plugins. Using the QGIS plugin installer, which itself is a Python plugin, it is possible to download a lot of existing Python plugins and to study their behaviour.

There is a collection of online documentation that may be usefull for PyQGIS programers:

- QGIS wiki: <http://wiki.qgis.org/qgiswiki/PythonBindings>
- QGIS API documentation: <http://doc.qgis.org/index.html>
- Qt documentation: <http://doc.trolltech.com/4.3/index.html>
- PyQt: <http://www.riverbankcomputing.co.uk/pyqt/>
- Python tutorial: <http://docs.python.org/>
- A book about desktop GIS and QGIS. It contains a chapter about PyQGIS plugin programming: <http://www.pragprog.com/titles/gsdgis/desktop-gis>

You can also write plugins for QGIS in C++. See Section 14 for more information about that.

16 Creating C++ Applications

Not everyone wants a full blown GIS desktop application. Sometimes you want to just have a widget inside your application that displays a map while the main goal of the application lies elsewhere. Perhaps a database frontend with a map display? This Section provides two simple code examples by Tim Sutton. They are available in the qgis subversion repository together with more interesting tutorials. Check out the whole repository from: https://svn.osgeo.org/qgis/trunk/code_examples/

16.1 Creating a simple mapping widget

With this first tutorial we take a little walk through creating a simple mapping widget. It won't do anything much - just load a shape file and display it in a random colour. But it should give you an idea of the potential for using QGIS as an embedded mapping component. Before we carry on, many thanks to Francis Bolduc who wrote the beginnings of this demo. He kindly agreed to make his work generally available.

We start with typical adding the necessary includes for our app:

```
//  
// QGIS Includes  
//  
#include <qgsapplication.h>  
#include <qgsproviderregistry.h>  
#include <qgssinglesymbolrenderer.h>  
#include <qgsmaplayerregistry.h>  
#include <qgsvectorlayer.h>  
#include <qgsmapcanvas.h>  
//  
// Qt Includes  
//  
#include <QString>  
#include <QApplication>  
#include <QWidget>
```

We use QgsApplication instead of Qt's QApplication, and get some added benefits of various static methods that can be used to locate library paths and so on.

The provider registry is a singleton that keeps track of vector data provider plugins. It does all the work for you of loading the plugins and so on. The single symbol renderer is the most basic symbology class. It renders points, lines or polygons in a single colour which is chosen at random by default (though you can set it yourself). Every vector layer must have a symbology associated with it.

The map layer registry keeps track of all the layers you are using. The vector layer class inherits from `maplayer` and extends it to include specialist functionality for vector data.

Finally the `mapcanvas` is really the nub of the matter. Its the drawable widget that our map will be drawn onto.

Now we can move on to initialising our application....

```
int main(int argc, char ** argv)
{
    // Start the Application
    QgsApplication app(argc, argv, true);

    QString myPluginsDir      = "/home/timlinux/apps/lib/qgis";
    QString myLayerPath       = "/home/timlinux/gisdata/brazil/BR_Cidades/";
    QString myLayerBaseName   = "Brasil_Cap";
    QString myProviderName    = "ogr";
```

So now we have a `qgsapplication` and we have defined some variables. Since I tested this on the Ubuntu 8.10, I just specified the location of the vector provider plugins as being inside the my development install directory. It would probaby make more sense in general to keep the QGIS libs in one of the standard library search paths on your system (e.g. `/usr/lib`) but this way will do for now.

The next two variables defined here just point to the shapefile I am going to be using (and you should substitute your own data here).

The provider name is important - it tells `qgis` which data provider to use to load the file. Typically you will use 'ogr' or 'postgres'.

Now we can go on to actually create our layer object.

```
// Instantiate Provider Registry
QgsProviderRegistry::instance(myPluginsDir);
```

First we get the provider registry initialised. Its a singleton class so we use the static instance call and pass it the provider lib search path. As it initialises it will scan this path for provider libs.

Now we go on to create a layer...

```
QgsVectorLayer * mypLayer =
    new QgsVectorLayer(myLayerPath, myLayerBaseName, myProviderName);
QgsSingleSymbolRenderer *mypRenderer = new
```

```
QgsSingleSymbolRenderer(mypLayer->geometryType());
    QList <QgsMapCanvasLayer> myLayerSet;

    mypLayer->setRenderer(mypRenderer);
    if (mypLayer->isValid())
    {
        qDebug("Layer is valid");
    }
    else
    {
        qDebug("Layer is NOT valid");
    }

    // Add the Vector Layer to the Layer Registry
    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer, TRUE);
    // Add the Layer to the Layer Set
    myLayerSet.append(QgsMapCanvasLayer(mypLayer, TRUE));
```

The code is fairly self explanatory here. We create a layer using the variables we defined earlier. Then we assign the layer a renderer. When we create a renderer, we need to specify the geometry type, which do do by asking the vector layer for its geometry type. Next we add the layer to a layerset (which is used by the QgsMapCanvas to keep track of which layers to render and in what order) and to the maplayer registry. Finally we make sure the layer will be visible.

Now we create a map canvas on to which we can draw the layer.

```
// Create the Map Canvas
QgsMapCanvas * mypMapCanvas = new QgsMapCanvas(0, 0);
mypMapCanvas->setExtent(mypLayer->extent());
mypMapCanvas->enableAntiAliasing(true);
mypMapCanvas->setCanvasColor(QColor(255, 255, 255));
mypMapCanvas->freeze(false);
// Set the Map Canvas Layer Set
mypMapCanvas->setLayerSet(myLayerSet);
mypMapCanvas->setVisible(true);
mypMapCanvas->refresh();
```

Once again there is nothing particularly tricky here. We create the canvas and then we set its extents to those of our layer. Next we tweak the canvas a bit to draw antialiased vectors. Next we set the background colour, unfreeze the canvas, make it visible and then refresh it.

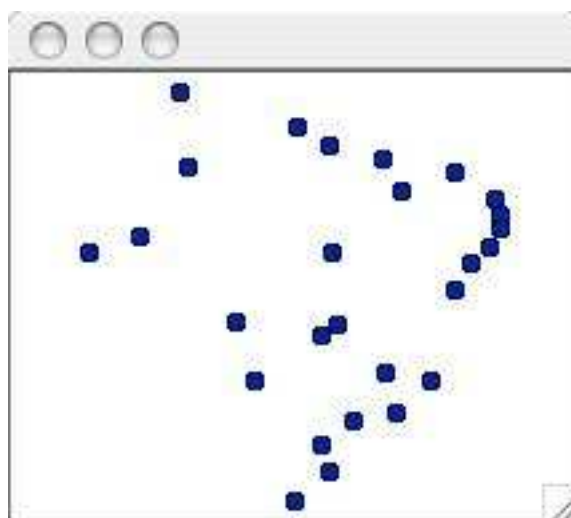
```
// Start the Application Event Loop
return app.exec();
}
```

In the last step we simply start the Qt event loop and we are all done. You can check out, compile and run this example using cmake like this:

```
svn co
https://svn.osgeo.org/qgis/trunk/code_examples/1_hello_world_qgis_style
cd 1_hello_world_qgis_style
mkdir build
#optionally specify where your QGIS is installed (should work on all
platforms)
#if your QGIS is installed to /usr or /usr/local you can leave this next step
out
export LIB_DIR=/home/timlinux/apps
cmake ..
make
./timtut1
```

When we compile and run it here is what the running app looks like:

Figure 58: Simple C++ Application **X**



16.2 Working with QgsMapCanvas

In Section 16.1 we showed you the usage of the QgsMapCanvas api to create a simple application that loads a shapefile and displays the points in it. But what good is a map that you can't interact with?

In this second tutorial I will extend the last tutorial by making it a QMainWindow application with a menu, toolbar and canvas area. We show you how to use QgsMapTool - the base class for all tools that need to interact with the map canvas. The purpose is to provide a demonstrator project, so I wont promise to write the most elegant or robust C++ code. The project will provide 4 toolbar icons for

- loading a map layer (layer name is hard coded in the application)
- zooming in
- zooming out
- panning

In the working directory for the tutorial code you will find a number of files including c++ sources, icons and a simple data file under data. There is also the .ui file for the main window.

Note: You will need to edit the .pro file in the above svn directory to match your system.

Since much of the code is the same as the previous tutorial, I will focus on the MapTool specifics - the rest of the implementation details can be investigated by checking out the project form SVN. A QgsMapTool is a class that interacts with the MapCanvas using the mouse pointer. QGIS has a number of QgsMapTools implemented, and you can subclass QgsMapTool to create your own. In mainwindow.cpp you will see I include the headers for the QgsMapTools near the start of the file:

```
//
// QGIS Map tools
//
#include "qgsmaptoolpan.h"
#include "qgsmaptoolzoom.h"
//
// These are the other headers for available map tools
// (not used in this example)
//
//#include "qgsmaptoolcapture.h"
//#include "qgsmaptoolidentify.h"
//#include "qgsmaptoolselect.h"
//#include "qgsmaptoolvertexedit.h"
//#include "qgsmeasure.h"
```

As you can see, I am only using two types of MapTool subclasses for this tutorial, but there are more available in the QGIS library. Hooking up our MapTools to the canvas is very easy using the normal Qt4 signal/slot mechanism:

```
//create the action behaviours
connect(mActionPan, SIGNAL(triggered()), this, SLOT(panMode()));
connect(mActionZoomIn, SIGNAL(triggered()), this, SLOT(zoomInMode()));
connect(mActionZoomOut, SIGNAL(triggered()), this, SLOT(zoomOutMode()));
connect(mActionAddLayer, SIGNAL(triggered()), this, SLOT(addLayer()));
```

Next we make a small toolbar to hold our toolbuttons. Note that the mpAction* actions were created in designer.

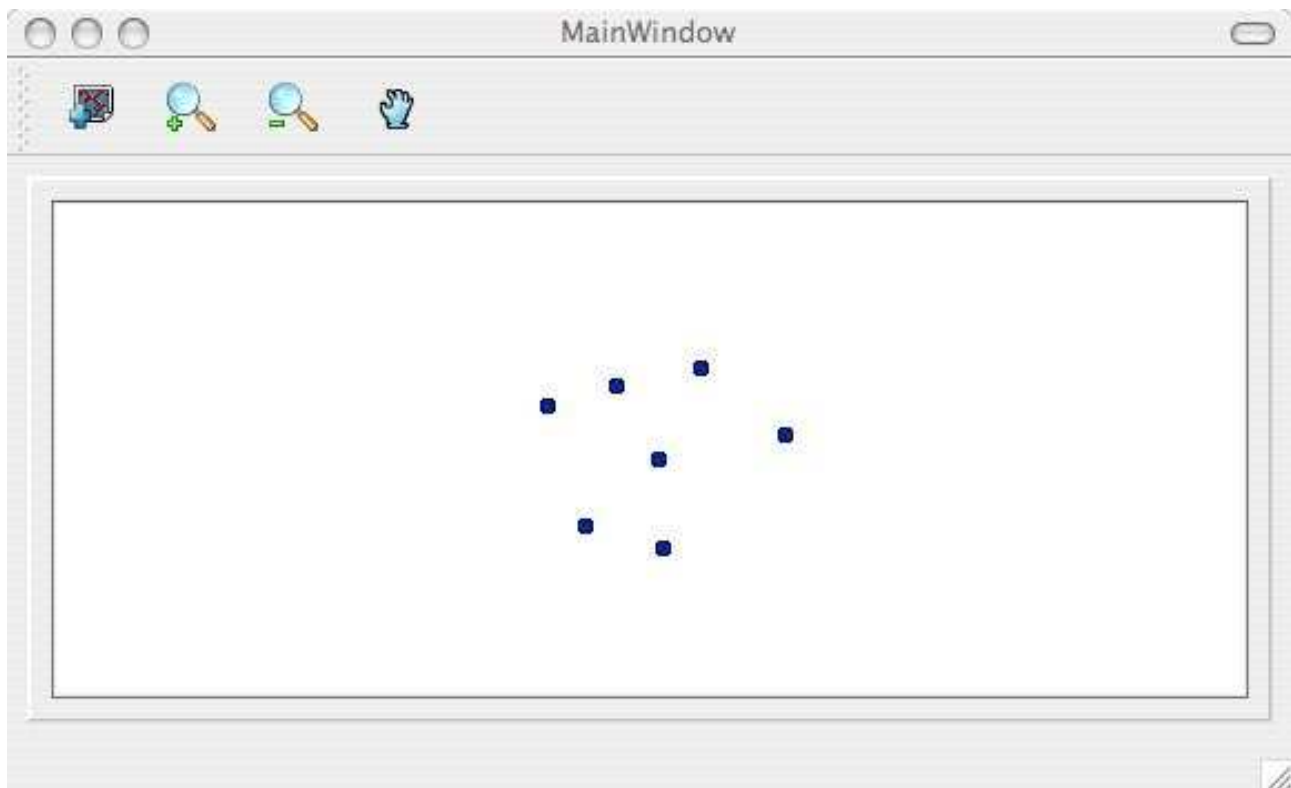
```
//create a little toolbar
mpMapToolBar = addToolBar(tr("File"));
mpMapToolBar->addAction(mpActionAddLayer);
mpMapToolBar->addAction(mpActionZoomIn);
mpMapToolBar->addAction(mpActionZoomOut);
mpMapToolBar->addAction(mpActionPan);
```

That's really pretty straightforward Qt stuff too. Now we create our three map tools:

```
//create the maptools
mpPanTool = new QgsMapToolPan(mpMapCanvas);
mpPanTool->setAction(mpActionPan);
mpZoomInTool = new QgsMapToolZoom(mpMapCanvas, FALSE); // false = in
mpZoomInTool->setAction(mpActionZoomIn);
mpZoomOutTool = new QgsMapToolZoom(mpMapCanvas, TRUE ); //true = out
mpZoomOutTool->setAction(mpActionZoomOut);
```

Again nothing here is very complicated - we are creating tool instances, each of which is associated with the same mapcanvas, and a different QAction. When the user selects one of the toolbar icons, the active MapTool for the canvas is set. For example when the pan icon is clicked, we do this:

```
void MainWindow::panMode()
{
    mpMapCanvas->setMapTool(mpPanTool);
}
```

Figure 59: QMainWindow application with a menu, toolbar and canvas area **X**

Conclusion

As you can see extending our previous example into something more functional using MapTools is really easy and only requires a few lines of code for each MapTool you want to provide.

You can check out and build this tutorial using SVN and CMake using the following steps:

```
svn co https://svn.osgeo.org/qgis/trunk/code_examples/2_basic_main_window
cd 2_basic_main_window
mkdir build
#optionally specify where your QGIS is installed (should work on all platforms)
#if your QGIS is installed to /usr or /usr/local you can leave this next step out
export LIB_DIR=/home/timlinux/apps
cmake ..
make
./timtut2
```

17 Creating PyQGIS Applications

One of the goals of QGIS is to provide not only an application, but a set of libraries that can be used to create new applications. This goal has been realized with the refactoring of libraries that took place after the release of 0.8. Since the release of 0.9, development of standalone applications using either C++ or Python is possible. We recommend you use QGIS 1.0.0 or greater as the basis for your python applications because since this version we now provide a stable consistent API.

In this chapter we'll take a brief look at the process for creating a standalone Python application. The QGIS blog has several examples of creating PyQGIS¹² applications. We'll use one of them as a starting point to get a look at how to create an application.

The features we want in the application are:

- Load a vector layer
- Pan
- Zoom in and out
- Zoom to the full extent of the layer
- Set custom colors when the layer is loaded

This is a pretty minimal feature set. Let's start by designing the GUI using Qt Designer.

17.1 Designing the GUI

Since we are creating a minimalistic application, we'll take the same approach with the GUI. Using Qt Designer, we create a simple MainWindow with no menu or toolbars. This gives us a blank slate to work with. To create the MainWindow:

1. Create a directory for developing the application and change to it
2. Run Qt Designer
3. The New Form dialog should appear. If it doesn't, choose New Form... from the File menu.
4. Choose Main Window from the templates/forms list
5. Click Create
6. Resize the new window to something manageable
7. Find the Frame widget in the list (under Containers) and drag it to the main window you just created
8. Click outside the frame to select the main window area

¹²An application created using Python and the QGIS bindings

9. Click on the Lay Out in a Grid tool. When you do, the frame will expand to fill your entire main window
10. Save the form as `mainwindow.ui`
11. Exit Qt Designer

Now compile the form using the PyQt interface compiler:

```
pyuic4 -o mainwindow_ui.py mainwindow.ui
```

This creates the Python source for the main window GUI. Next we need to create the application code to fill the blank slate with some tools we can use.

17.2 Creating the MainWindow

Now we are ready to write the **MainWindow** class that will do the real work. Since it takes up quite a few lines, we'll look at it in chunks, starting with the import section and environment setup:

```
1 # Loosely based on:
2 #   Original C++ Tutorial 2 by Tim Sutton
3 #   ported to Python by Martin Dobias
4 #   with enhancements by Gary Sherman for FOSS4G2007
5 # Licensed under the terms of GNU GPL 2
6
7 from PyQt4.QtCore import *
8 from PyQt4.QtGui import *
9 from qgis.core import *
10 from qgis.gui import *
11 import sys
12 import os
13 # Import our GUI
14 from mainwindow_ui import Ui_MainWindow
15
16 # Environment variable QGISHOME must be set to the 1.0 install directory
17 # before running this application
18 qgis_prefix = os.getenv("QGISHOME")
```

Some of this should look familiar from our plugin, especially the PyQt4 and QGIS imports. Some specific things to note are the import of our GUI in line 14 and the import of our CORE library on line 9.

Our application needs to know where to find the QGIS installation. Because of this, we set the QGISHOME environment variable to point to the install directory of QGIS 1.x In line 20 we store this value from the environment for later use.

Next we need to create our **MainWindow** class which will contain all the logic of our application.

```
21 class MainWindow(QMainWindow, Ui_MainWindow):
22
23     def __init__(self):
24         QMainWindow.__init__(self)
25
26         # Required by Qt4 to initialize the UI
27         self.setupUi(self)
28
29         # Set the title for the app
30         self.setWindowTitle("QGIS Demo App")
31
32         # Create the map canvas
33         self.canvas = QgsMapCanvas()
34         # Set the background color to light blue something
35         self.canvas.setCanvasColor(QColor(200,200,255))
36         self.canvas.enableAntiAliasing(True)
37         self.canvas.useQImageToRender(False)
38         self.canvas.show()
39
40         # Lay our widgets out in the main window using a
41         # vertical box layout
42         self.layout = QVBoxLayout(self.frame)
43         self.layout.addWidget(self.canvas)
44
45         # Create the actions for our tools and connect each to the appropriate
46         # method
47         self.actionAddLayer = QAction(QIcon("(qgis_prefix + "/share/qgis/themes/classic/mActionZ
48         \
49         "Add Layer", self.frame)
50         self.connect(self.actionAddLayer, SIGNAL("activated()"), self.addLayer)
51         self.actionZoomIn = QAction(QIcon("(qgis_prefix + "/share/qgis/themes/classic/mActionZ
52         "Zoom In", self.frame)
53         self.connect(self.actionZoomIn, SIGNAL("activated()"), self.zoomIn)
54         self.actionZoomOut = QAction(QIcon("(qgis_prefix + "/share/qgis/themes/classic/mActionZ
55         "Zoom Out", self.frame)
56         self.connect(self.actionZoomOut, SIGNAL("activated()"), self.zoomOut)
```

```
57     self.actionPan = QAction(QIcon("(qgis_prefix + "/share/qgis/themes/classic/mActionPan.pn
58         "Pan", self.frame)
59     self.connect(self.actionPan, SIGNAL("activated()"), self.pan)
60     self.actionZoomFull = QAction(QIcon("(qgis_prefix + "/share/qgis/themes/classic/mActionZ
61         "Zoom Full Extent", self.frame)
62     self.connect(self.actionZoomFull, SIGNAL("activated()"),
63         self.zoomFull)
64
65     # Create a toolbar
66     self.toolbar = self.addToolBar("Map")
67     # Add the actions to the toolbar
68     self.toolbar.addAction(self.actionAddLayer)
69     self.toolbar.addAction(self.actionZoomIn)
70     self.toolbar.addAction(self.actionZoomOut);
71     self.toolbar.addAction(self.actionPan);
72     self.toolbar.addAction(self.actionZoomFull);
73
74     # Create the map tools
75     self.toolPan = QgsMapToolPan(self.canvas)
76     self.toolZoomIn = QgsMapToolZoom(self.canvas, False) # false = in
77     self.toolZoomOut = QgsMapToolZoom(self.canvas, True) # true = out
```

Lines 21 through 27 are the basic declaration and initialization of the **MainWindow** and the set up of the user interface using the *setupUi* method. This is required for all applications.

Next we set the title for the application so it says something more interesting than `MainWindow` (line 30). Once that is complete, we are ready to complete the user interface. When we created it in Designer, we left it very sparse—just a main window and a frame. You could have added a menu and the toolbar using Designer, however we'll do it with Python.

In lines 33 through 38 we set up the map canvas, set the background color to a light blue, and enable antialiasing. We also tell it not to use a **QImage** for rendering (trust me on this one) and then set the canvas to visible by calling the *show* method.

Next we set the layer to use a vertical box layout within the frame and add the map canvas to it in line 43.

Lines 48 to 63 set up the actions and connections for the tools in our toolbar. For each tool, we create a **QAction** using the icon we defined in the QGIS classic theme. Then we connect up the `activated` signal from the tool to the method in our class that will handle the action. This is similar to how we set things up in the plugin example.

Once we have the actions and connections, we need to add them to the toolbar. In lines 66 through 72 we create the toolbar and add each tool to it.

Lastly we create the three map tools for the application (lines 75 through 77). We'll use the map tools in a moment when we define the methods to make our application functional. Let's look at the methods for the map tools.

```
78 # Set the map tool to zoom in
79 def zoomIn(self):
80     self.canvas.setMapTool(self.toolZoomIn)
81
82 # Set the map tool to zoom out
83 def zoomOut(self):
84     self.canvas.setMapTool(self.toolZoomOut)
85
86 # Set the map tool to
87 def pan(self):
88     self.canvas.setMapTool(self.toolPan)
89
90 # Zoom to full extent of layer
91 def zoomFull(self):
92     self.canvas.zoomFullExtent()
```

For each map tool, we need a method that corresponds to the connection we made for each action. In lines 79 through 88 we set up a method for each of the three tools that interact with the map. When a tool is activated by clicking on it in the toolbar, the corresponding method is called that “tells” the map canvas it is the active tool. The active tool governs what happens when the mouse is clicked on the canvas.

The zoom to full extent tool isn't a map tool—it does its job without requiring a click on the map. When it is activated, we call the *zoomFullExtent* method of the map canvas (line 92). This completes the implementation of all our tools except one—the Add Layer tool. Let's look at it next:

```
93 # Add an OGR layer to the map
94 def addLayer(self):
95     file = QFileDialog.getOpenFileName(self, "Open Shapefile", ".", "Shapefiles
96     (*.shp)")
97     fileInfo = QFileInfo(file)
98
99     # Add the layer
100     layer = QgsVectorLayer(file, fileInfo.fileName(), "ogr")
101
102     if not layer.isValid():
103         return
104
```

```
105     # Change the color of the layer to gray
106     symbols = layer.renderer().symbols()
107     symbol = symbols[0]
108     symbol.setFill(QColor.fromRgb(192,192,192))
109
110     # Add layer to the registry
111     QgsMapLayerRegistry.instance().addMapLayer(layer);
112
113     # Set extent to the extent of our layer
114     self.canvas.setExtent(layer.extent())
115
116     # Set up the map canvas layer set
117     cl = QgsMapCanvasLayer(layer)
118     layers = [cl]
119     self.canvas.setLayerSet(layers)
```

In the `addLayer` method we use a **QFileDialog** to get the name of the shapefile to load. This is done in line 96. Notice that we specify a “filter” so the dialog will only show files of type `.shp`.

Next in line 97 we create a **QFileInfo** object from the shapefile path. Now the layer is ready to be created in line 100. Using the **QFileInfo** object to get the file name from the path we specify it for the name of the layer when it is created. To make sure that the layer is valid and won’t cause any problems when loading, we check it in line 102. If it’s bad, we bail out and don’t add it to the map canvas.

Normally layers are added with a random color. Here we want to tweak the colors for the layer to make a more pleasing display. Plus we know we are going to add the `world_borders` layer to the map and this will make it look nice on our blue background. To change the color, we need to get the symbol used for rendering and use it to set a new fill color. This is done in lines 106 through 108.

All that’s left is to actually add the layer to the registry and a few other housekeeping items (lines 111 through 119). This stuff is standard for adding a layer and the end result is the world borders on a light blue background. The only thing you may not want to do is set the extent to the layer, if you are going to be adding more than one layer in your application.

That’s the heart of the application and completes the **MainWindow** class.

17.3 Finishing Up

The remainder of the code shown below creates the *QgsApplication* object, sets the path to the QGIS install, sets up the *main* method and then starts the application. The only other thing to note is that we move the application window to the upper left of the display. We could get fancy and use the Qt

API to center it on the screen.

```
120 def main(argv):
121     # create Qt application
122     app = QApplication(argv)
123
124     # Initialize qgis libraries
125     QgsApplication.setPrefixPath(qgis_prefix, True)
126     QgsApplication.initQgis()
127
128     # create main window
129     wnd = MainWindow()
130     # Move the app window to upper left
131     wnd.move(100,100)
132     wnd.show()
133
134     # run!
135     retval = app.exec_()
136
137     # exit
138     QgsApplication.exitQgis()
139     sys.exit(retval)
140
141
142 if __name__ == "__main__":
143     main(sys.argv)
```

17.4 Running the Application

Now we can run the application and see what happens. Of course if you are like most developers, you've been testing it out as you went along.

Before we can run the application, we need to set some environment variables.





```
export LD_LIBRARY_PATH=$HOME/qgis/lib%$
export PYTHONPATH=$HOME/qgis/share/qgis/python
export QGISHOME=$HOME/qgis%$
```



```
set PATH=C:\qgis;%PATH%
set PYTHONPATH=C:\qgis\python
set QGISHOME=C:\qgis
```

We assume

-  QGIS is installed in your home directory in qgis.
-  QGIS is installed in C:\qgis.

When the application starts up, it looks like this:

To add the `world_borders` layer, click on the `Add Layer` tool and navigate to the data directory. Select the shapefile and click to add it to the map. Our custom fill color is applied and the result is:

Creating a PyQGIS application is really pretty simple. In less than 150 lines of code we have an application that can load a shapefile and navigate the map. If you play around with the map, you'll notice that some of the built-in features of the canvas also work, including mouse wheel scrolling and panning by holding down the bar and moving the mouse.

Some sophisticated applications have been created with PyQGIS and more are in the works. This is pretty impressive, considering that this development has taken place even before the official release of QGIS 1.0.

Tip 45 DOCUMENTATION FOR PYQGIS

Whether you are writing a plugin or a PyQGIS application, you are going to need to refer to both the QGIS API documentation (<http://doc.qgis.org>) and the PyQt Python Bindings Reference Guide (<http://www.riverbankcomputing.com/Docs/PyQt4/pyqt4ref.html>). These documents provide information about the classes and methods you'll use to bring your Python creation to life.

18 Help and Support

18.1 Mailinglists

QGIS is under active development and as such it won't always work like you expect it to. The preferred way to get help is by joining the qgis-users mailing list.

qgis-users

Your questions will reach a broader audience and answers will benefit others. You can subscribe to the qgis-users mailing list by visiting the following URL:

<http://lists.osgeo.org/mailman/listinfo/qgis-user>

qgis-developer

If you are a developer facing problems of a more technical nature, you may want to join the qgis-developer mailing list here:

<http://lists.osgeo.org/mailman/listinfo/qgis-developer>

qgis-commit

Each time a commit is made to the QGIS code repository an email is posted to this list. If you want to be up to date with every change to the current code base, you can subscribe to this list at:

<http://lists.osgeo.org/mailman/listinfo/qgis-commit>

qgis-trac

This list provides email notification related to project management, including bug reports, tasks, and feature requests. You can subscribe to this list at:

<http://lists.osgeo.org/mailman/listinfo/qgis-trac>

qgis-community-team

This list deals with topics like documentation, context help, user-guide, online experience including web sites, blog, mailing lists, forums, and translation efforts. If you like to work on the user-guide as well, this list is a good starting point to ask your questions. You can subscribe to this list at:

<http://lists.osgeo.org/mailman/listinfo/qgis-community-team>

qgis-release-team

This list deals with topics like the release process, packaging binaries for various OS and announcing new releases to the world at large. You can subscribe to this list at:

<http://lists.osgeo.org/mailman/listinfo/qgis-release-team>

qgis-psc

This list is used to discuss Steering Committee issues related to overall management and direction of Quantum GIS. You can subscribe to this list at:

<http://lists.osgeo.org/mailman/listinfo/qgis-psc>

You are welcome to subscribe to any of the lists. Please remember to contribute to the list by answering questions and sharing your experiences. Note that the qgis-commit and qgis-trac are designed for notification only and not meant for user postings.

18.2 IRC

We also maintain a presence on IRC - visit us by joining the #qgis channel on irc.freenode.net. Please wait around for a response to your question as many folks on the channel are doing other things and it may take a while for them to notice your question. Commercial support for QGIS is also available. Check the website <http://qgis.org/content/view/90/91> for more information.

If you missed a discussion on IRC, not a problem! We log all discussion so you can easily catch up. Just go to <http://logs.qgis.org> and read the IRC-logs.

18.3 BugTracker

While the qgis-users mailing list is useful for general 'how do I do xyz in QGIS' type questions, you may wish to notify us about bugs in QGIS. You can submit bug reports using the QGIS bug tracker at <https://trac.osgeo.org/qgis/>. When creating a new ticket for a bug, please provide an email address where we can request additional information.

Please bear in mind that your bug may not always enjoy the priority you might hope for (depending on its severity). Some bugs may require significant developer effort to remedy and the manpower is not always available for this.

Feature requests can be submitted as well using the same ticket system as for bugs. Please make sure to select the type `enhancement`.

If you have found a bug and fixed it yourself you can submit this patch also. Again, the lovely trac ticketsystem at <https://trac.osgeo.org/qgis/> has this type as well. Select `patch` from the type-menu. Someone of the developers will review it and apply it to QGIS.

Please don't be alarmed if your patch is not applied straight away - developers may be tied up with other commitments.

18.4 Blog

The QGIS-community also runs a weblog (BLOG) at <http://blog.qgis.org> which has some interesting articles for users and developers as well. You are invited to contribute to the blog after registering yourself!

18.5 Wiki

Lastly, we maintain a WIKI web site at <http://wiki.qgis.org> where you can find a variety of useful information relating to QGIS development, release plans, links to download sites, message translation-hints and so on. Check it out, there are some goodies inside!

A Supported Data Formats

A.1 Supported OGR Formats

At the date of this document, the following formats are supported by the OGR library. Formats known to work in QGIS are indicated in **bold**.

- **Arc/Info Binary Coverage**
- Comma Separated Value (.csv)
- DODS/OPeNDAP
- **ESRI Shapefile**
- FMEObjects Gateway
- GML
- IHO S-57 (ENC)
- **Mapinfo File**
- Microstation DGN
- OGDl Vectors
- ODBC
- Oracle Spatial
- PostgreSQL¹³
- **SDTS**
- SQLite
- UK .NTF
- U.S. Census TIGER/Line
- VRT - Virtual Datasource

A.2 GDAL Raster Formats

At the date of this document, the following formats are supported by the GDAL library. Note that not all of these format may work in QGIS for various reasons. For example, some require external commercial libraries. Only those formats that have been well tested will appear in the list of file types when loading a raster into QGIS. Other untested formats can be loaded by selecting the *All other files (*)* filter. Formats known to work in QGIS are indicated in **bold**.

¹³QGIS implements its own PostgreSQL functions. OGR should be built without PostgreSQL support

- **Arc/Info ASCII Grid**
- **Arc/Info Binary Grid (.adf)**
- Microsoft Windows Device Independent Bitmap (.bmp)
- BSB Nautical Chart Format (.kap)
- VTP Binary Terrain Format (.bt)
- CEOS (Spot for instance)
- First Generation USGS DOQ (.doq)
- New Labelled USGS DOQ (.doq)
- Military Elevation Data (.dt0, .dt1)
- ERMapper Compressed Wavelets (.ecw)
- ESRI .hdr Labelled
- ENVI .hdr Labelled Raster
- Envisat Image Product (.n1)
- EOSAT FAST Format
- FITS (.fits)
- Graphics Interchange Format (.gif)
- **GRASS Rasters**¹⁴
- **TIFF / GeoTIFF (.tif)**
- Hierarchical Data Format Release 4 (HDF4)
- **Erdas Imagine (.img)**
- Atlantis MFF2e
- Japanese DEM (.mem)
- **JPEG JFIF (.jpg)**
- JPEG2000 (.jp2, .j2k)
- JPEG2000 (.jp2, .j2k)
- NOAA Polar Orbiter Level 1b Data Set (AVHRR)
- Erdas 7.x .LAN and .GIS
- In Memory Raster
- Atlantis MFF
- Multi-resolution Seamless Image Database MrSID
- NITF

¹⁴GRASS raster support is supplied by the QGIS GRASS data provider plugin

A SUPPORTED DATA FORMATS

- NetCDF
- OGD Bridge
- PCI .aux Labelled
- PCI Geomatics Database File
- Portable Network Graphics (.png)
- Netpbm (.ppm,.pgm)
- **USGS SDTS DEM (*CATD.DDF)**
- SAR CEOS
- **USGS ASCII DEM (.dem)**
- X11 Pixmap (.xpm)

B GRASS Toolbox modules

The GRASS Shell inside the GRASS Toolbox provides access to almost all (more than 300) GRASS modules in command line modus. To offer a more user friendly working environment, about 200 of the available GRASS modules and functionalities are also provided by graphical dialogs.

B.1 GRASS Toolbox data import and export modules

This Section lists all graphical dialogs in the GRASS Toolbox to import and export data into a currently selected GRASS location and mapset.

Table 8: GRASS Toolbox: Data import modules

Data import modules in the GRASS Toolbox	
Module name	Purpose
r.in.arc	Convert an ESRI ARC/INFO ascii raster file (GRID) into a (binary) raster map layer
r.in.ascii	Convert an ASCII raster text file into a (binary) raster map layer
r.in.aster	Georeferencing, rectification, and import of Terra-ASTER imagery and relative DEM's using gdalwarp
r.in.gdal	Import GDAL supported raster file into a GRASS binary raster map layer
r.in.gdal.loc	Import GDAL supported raster file into a GRASS binary raster map layer and create a fitted location
r.in.gridatb	Imports GRIDATB.FOR map file (TOPMODEL) into GRASS raster map
r.in.mat	Import a binary MAT-File(v4) to a GRASS raster
r.in.poly	Create raster maps from ascii polygon/line data files in the current directory
r.in.srtm	Import SRTM HGT files into GRASS
i.in.spotvgt	Import of SPOT VGT NDVI file into a raster map
v.in.dxf	Import DXF vector layer
v.in.e00	Import ESRI E00 file in a vector map
v.in.garmin	Import vector from gps using gpstrans
v.in.gpsbabel	Import vector from gps using gpsbabel
v.in.mapgen	Import MapGen or MatLab vectors in GRASS
v.in.ogr	Import OGR/PostGIS vector layers
v.in.ogr.loc	Import OGR/PostGIS vector layers and create a fitted location
v.in.ogr.all	Import all the OGR/PostGIS vector layers in a given data source
v.in.ogr.all.loc	Import all the OGR/PostGIS vector layers in a given data source and create a fitted location

Table 9: GRASS Toolbox: Data export modules

Data export modules in the GRASS Toolbox	
Module name	Purpose
r.out.gdal.gtiff	Export raster layer to Geo TIFF
r.out.arc	Converts a raster map layer into an ESRI ARCGRID file
r.gridatb	Exports GRASS raster map to GRIDATB.FOR map file (TOPMODEL)
r.out.mat	Exports a GRASS raster to a binary MAT-File
r.out.bin	Exports a GRASS raster to a binary array
r.out.png	Export GRASS raster as non-georeferenced PNG image format
r.out.ppm	Converts a GRASS raster map to a PPM image file at the pixel resolution of the CURRENTLY DEFINED REGION
r.out.ppm3	Converts 3 GRASS raster layers (R,G,B) to a PPM image file at the pixel resolution of the CURRENTLY DEFINED REGION
r.out.pov	Converts a raster map layer into a height-field file for POVRAY
r.out.tiff	Exports a GRASS raster map to a 8/24bit TIFF image file at the pixel resolution of the currently defined region
r.out.vrml	Export a raster map to the Virtual Reality Modeling Language (VRML)
v.out.ogr	Export vector layer to various formats (OGR library)
v.out.ogr.gml	Export vector layer to GML
v.out.ogr.postgis	Export vector layer to various formats (OGR library)
v.out.ogr.mapinfo	Mapinfo export of vector layer
v.out.ascii	Convert a GRASS binary vector map to a GRASS ASCII vector map
v.out.dxf	converts a GRASS vector map to DXF

B.2 GRASS Toolbox data type conversion modules

This Section lists all graphical dialogs in the GRASS Toolbox to convert raster to vector or vector to raster data in a currently selected GRASS location and mapset.

Table 10: GRASS Toolbox: Data type conversion modules

Data type conversion modules in the GRASS Toolbox	
Module name	Purpose
r.to.vect.point	Convert a raster to vector points
r.to.vect.line	Convert a raster to vector lines
r.to.vect.area	Convert a raster to vector areas
v.to.rast.constant	Convert a vector to raster using constant
v.to.rast.attr	Convert a vector to raster using attribute values

B.3 GRASS Toolbox region and projection configuration modules

This Section lists all graphical dialogs in the GRASS Toolbox to manage and change the current mapset region and to configure your projection.

Table 11: GRASS Toolbox: Region and projection configuration modules

Region and projection configuration modules in the GRASS Toolbox	
Module name	Purpose
g.region.save	Save the current region as a named region
g.region.zoom	Shrink the current region until it meets non-NULL data from a given raster map
g.region.multiple.raster	Set the region to match multiple raster maps
g.region.multiple.vector	Set the region to match multiple vector maps
g.proj.print	Print projection information of the current location
g.proj.geo	Print projection information from a georeferenced file (raster, vector or image)
g.proj.ascii.new	Print projection information from a georeferenced ASCII file containing a WKT projection description
g.proj.proj	Print projection information from a PROJ.4 projection description file
g.proj.ascii.new	Print projection information from a georeferenced ASCII file containing a WKT projection description and create a new location based on it
g.proj.geo.new	Print projection information from a georeferenced file (raster, vector or image) and create a new location based on it
g.proj.proj.new	Print projection information from a PROJ.4 projection description file and create a new location based on it
m.cogo	A simple utility for converting bearing and distance measurements to coordinates and vice versa. It assumes a cartesian coordinate system

B.4 GRASS Toolbox raster data modules

This Section lists all graphical dialogs in the GRASS Toolbox to work with and analyse raster data in a currently selected GRASS location and mapset.

Table 12: GRASS Toolbox: Develop raster map modules

Develop raster map modules in the GRASS Toolbox	
Module name	Purpose
r.compress	Compresses and decompresses raster maps
r.region.region	Sets the boundary definitions to current or default region
r.region.raster	Sets the boundary definitions from existent raster map
r.region.vector	Sets the boundary definitions from existent vector map
r.region.edge	Sets the boundary definitions by edge (n-s-e-w)
r.region.alignTo	Sets region to align to a raster map
r.null.val	Transform cells with value in null cells
r.null.to	Transform null cells in value cells
r.quant	This routine produces the quantization file for a floating-point map
r.resamp.stats	Resamples raster map layers using aggregation
r.resamp.interp	Resamples raster map layers using interpolation
r.resample	GRASS raster map layer data resampling capability. Before you must set new resolution
r.resamp.rst	Reinterpolates and computes topographic analysis using regularized spline with tension and smoothing
r.support	Allows creation and/or modification of raster map layer support files
r.support.stats	Update raster map statistics
r.proj	Re-project a raster map from one location to the current location

Table 13: GRASS Toolbox: Raster color management modules

Raster color management modules in the GRASS Toolbox	
Module name	Purpose
r.colors.table	Set raster color table from setted tables
r.colors.rules	Set raster color table from setted rules
r.colors.rast	Set raster color table from existing raster
r.blend	Blend color components for two raster maps by given ratio
r.composite	Blend red, green, raster layers to obtain one color raster
r.his	Generates red, green and blue raster map layers combining hue, intensity, and saturation (his) values from user-specified input raster map layers

Table 14: GRASS Toolbox: Spatial raster analysis modules

Spatial raster analysis modules in the GRASS Toolbox	
Module name	Purpose
r.buffer	Raster buffer
r.mask	Create a MASK for limiting raster operation
r.mapcalc	Raster map calculator
r.mapcalculator	Simple map algebra
r.neighbors	Raster neighbors analyses
v.neighbors	Count of neighbouring points
r.cross	Create a cross product of the category value from multiple raster map layers
r.series	Makes each output cell a function of the values assigned to the corresponding cells in the output raster map layers
r.patch	Create a new raster map by combining other raster maps
r.statistics	Category or object oriented statistics
r.cost	Outputs a raster map layer showing the cumulative cost of moving between different geographic locations on an input raster map layer whose cell category values represent cost
r.drain	Traces a flow through an elevation model on a raster map layer
r.shaded.relief	Create shaded map
r.slope.aspect.slope	Generate slope map from DEM (digital elevation model)
r.slope.aspect.aspect	Generate aspect map from DEM (digital elevation model)
r.param.scale	Extracts terrain parameters from a DEM
r.texture	Generate images with textural features from a raster map (first serie of indices)
r.texture.bis	Generate images with textural features from a raster map (second serie of indices)
r.los	Line-of-sigth raster analysis
r.clump	Recategorizes into unique categories contiguous cells
r.grow	Generates a raster map layer with contiguous areas grown by one cell
r.thin	Thin no-zero cells that denote line features

Table 15: GRASS Toolbox: Surface management modules

Surface management modules in the GRASS Toolbox	
Module name	Purpose
r.random	Creates a random vector point map contained in a raster
r.random.cells	Generates random cell values with spatial dependence
v.kernel	Gaussian kernel density
r.contour	Produces a contours vector map with specified step from a raster map
r.contour2	Produces a contours vector map of specified contours from a raster map
r.surf.fractal	Creates a fractal surface of a given fractal dimension
r.surf.gauss	GRASS module to produce a raster map layer of gaussian deviates whose mean and standard deviation can be expressed by the user
r.surf.random	Produces a raster map layer of uniform random deviates whose range can be expressed by the user
r.bilinear	Bilinear interpolation utility for raster map layers
v.surf.bispline	Bicubic or bilinear spline interpolation with Tykhonov regularization
r.surf.idw	Surface interpolation utility for raster map layers
r.surf.idw2	Surface generation program
r.surf.contour	Surface generation program from rasterized contours
v.surf.idw	Interpolate attribute values (IDW)
v.surf.rst	Interpolate attribute values (RST)
r.fillnulls	Fills no-data areas in raster maps using v.surf.rst splines interpolation

Table 16: GRASS Toolbox: Change raster category values and labels modules

Raster category and label modules in the GRASS Toolbox	
Module name	Purpose
r.reclass.area.greater	Reclasses a raster map greater than user specified area size (in hectares)
r.reclass.area.lessor	Reclasses a raster map less than user specified area size (in hectares)
r.reclass	Reclass a raster using a reclassification rules file
r.recode	Recode raster maps
r.rescale	Rescales the range of category values in a raster map layer

Table 17: GRASS Toolbox: Hydrologic modelling modules

Hydrologic modelling modules in the GRASS Toolbox	
Module name	Purpose
r.carve	Takes vector stream data, transforms it to raster, and subtracts depth from the output DEM
r.fill.dir	Filters and generates a depressionless elevation map and a flow direction map from a given elevation layer
r.lake.xy	Fills lake from seed point at given level
r.lake.seed	Fills lake from seed at given level
r.topidx	Creates a 3D volume map based on 2D elevation and value raster maps
r.basins.fill	Generates a raster map layer showing watershed subbasins
r.water.outlet	Watershed basin creation program

Table 18: GRASS Toolbox: Reports and statistic analysis modules

Reports and statistic analysis modules in the GRASS Toolbox	
Module name	Purpose
r.category	Prints category values and labels associated with user-specified raster map layers
r.sum	Sums up the raster cell values
r.report	Reports statistics for raster map layers
r.average	Finds the average of values in a cover map within areas assigned the same category value in a user-specified base map
r.median	Finds the median of values in a cover map within areas assigned the same category value in a user-specified base map
r.mode	Finds the mode of values in a cover map within areas assigned the same category value in a user-specified base map.reproject raster image
r.volume	Calculates the volume of data clumps, and produces a GRASS vector points map containing the calculated centroids of these clumps
r.surf.area	Surface area estimation for rasters
r.univar	Calculates univariate statistics from the non-null cells of a raster map
r.covar	Outputs a covariance/correlation matrix for user-specified raster map layer(s)
r.regression.line	Calculates linear regression from two raster maps: $y = a + b * x$
r.coin	Tabulates the mutual occurrence (coincidence) of categories for two raster map layers

B.5 GRASS Toolbox vector data modules

This Section lists all graphical dialogs in the GRASS Toolbox to work with and analyse vector data in a currently selected GRASS location and mapset.

Table 19: GRASS Toolbox: Develop vector map modules

Develop vector map modules in the GRASS Toolbox	
Module name	Purpose
v.build.all	Rebuild topology of all vectors in the mapset
v.clean.break	Break lines at each intersection of vector map
v.clean.snap	Cleaning topology: snap lines to vertex in threshold
v.clean.rmdangles	Cleaning topology: remove dangles
v.clean.chdangles	Cleaning topology: change the type of boundary dangle to line
v.clean.rmbridge	Remove bridges connecting area and island or 2 islands
v.clean.chbridge	Change the type of bridges connecting area and island or 2 islands
v.clean.rmdupl	Remove duplicate lines (pay attention to categories!)
v.clean.rmdac	Remove duplicate area centroids
v.clean.bpol	Break polygons. Boundaries are broken on each point shared between 2 and more polygons where angles of segments are different
v.clean.prune	Remove vertices in threshold from lines and boundaries
v.clean.rmarea	Remove small areas (removes longest boundary with adjacent area)
v.clean.rmline	Remove all lines or boundaries of zero length
v.clean.rmsa	Remove small angles between lines at nodes
v.type.lb	Convert lines to boundaries
v.type.bl	Convert boundaries to lines
v.type.pc	Convert points to centroids
v.type.cp	Convert centroids to points
v.centroids	Add missing centroids to closed boundaries
v.build.polylines	Build polylines from lines
v.segment	Creates points/segments from input vector lines and positions
v.to.points	Create points along input lines
v.parallel	Create parallel line to input lines
v.dissolve	Dissolves boundaries between adjacent areas
v.drape	Convert 2D vector to 3D vector by sampling of elevation raster
v.transform	Performs an affine transformation on a vector map
v.proj	Allows projection conversion of vector files
v.support	Updates vector map metadata
generalize	Vector based generalization

Table 20: GRASS Toolbox: Database connection modules

Database connection modules in the GRASS Toolbox	
Module name	Purpose
v.db.connect	Connect a vector to database
v.db.sconnect	Disconnect a vector from database
v.db.what.connect	Set/Show database connection for a vector

Table 21: GRASS Toolbox: Change vector field modules

Change vector field modules in the GRASS Toolbox	
Module name	Purpose
v.category.add	Add elements to layer (ALL elements of the selected layer type!)
v.category.del	Delete category values
v.category.sum	Add a value to the current category values
v.reclass.file	Reclass category values using a rules file
v.reclass.attr	Reclass category values using a column attribute (integer positive)

Table 22: GRASS Toolbox: Working with vector points modules

Working with vector points modules in the GRASS Toolbox	
Module name	Purpose
v.in.region	Create new vector area map with current region extent
v.mkgrid.region	Create grid in current region
v.in.db	Import vector points from a database table containing coordinates
v.random	Randomly generate a 2D/3D GRASS vector point map
v.kcv	Randomly partition points into test/train sets
v.outlier	Remove outliers from vector point data
v.hull	Create a convex hull
v.delaunay.line	Delaunay triangulation (lines)
v.delaunay.area	Delaunay triangulation (areas)
v.voronoi.line	Voronoi diagram (lines)
v.voronoi.area	Voronoi diagram (areas)

Table 23: GRASS Toolbox: Spatial vector and network analysis modules

Spatial vector and network analysis modules in the GRASS Toolbox	
Module name	Purpose
v.extract.where	Select features by attributes
v.extract.list	Extract selected features
v.select.overlap	Select features overlapped by features in another map
v.buffer	Vector buffer
v.distance	Find the nearest element in vector 'to' for elements in vector 'from'.
v.net.nodes	Create nodes on network
v.net.alloc	Allocate network
v.net.iso	Cut network by cost isolines
v.net.salesman	Connect nodes by shortest route (traveling salesman)
v.net.steiner	Connect selected nodes by shortest tree (Steiner tree)
v.patch	Create a new vector map by combining other vector maps
v.overlay.or	Vector union
v.overlay.and	Vector intersection
v.overlay.not	Vector subtraction
v.overlay.xor	Vector non-intersection

Table 24: GRASS Toolbox: Vector update by other maps modules

Vector update by other maps modules in the GRASS Toolbox	
Module name	Purpose
v.rast.stats	Calculates univariate statistics from a GRASS raster map based on vector objects
v.what.vect	Uploads map for which to edit attribute table
v.what.rast	Uploads raster values at positions of vector points to the table
v.sample	Sample a raster file at site locations

Table 25: GRASS Toolbox: Vector report and statistic modules

Vector report and statistic modules in the GRASS Toolbox	
Module name	Purpose
v.to.db	Put geometry variables in database
v.report	Reports geometry statistics for vectors
v.univar	Calculates univariate statistics on selected table column for a GRASS vector map
v.normal	Tests for normality for points

B.6 GRASS Toolbox imagery data modules

This Section lists all graphical dialogs in the GRASS Toolbox to work with and analyse imagery data in a currently selected GRASS location and mapset.

Table 26: GRASS Toolbox: Imagery analysis modules

Imagery analysis modules in the GRASS Toolbox	
Module name	Purpose
i.image.mosaik	Mosaic up to 4 images
i.rgb.his	Red Green Blue (RGB) to Hue Intensity Saturation (HIS) raster map color transformation function
i.his.rgb	Hue Intensity Saturation (HIS) to Red Green Blue (RGB) raster map color transform function
i.landsat.rgb	Auto-balancing of colors for LANDSAT images
i.fusion.brovey	Brovey transform to merge multispectral and high-res pancromatic channels
i.zc	Zero-crossing edge detection raster function for image processing
i.mfilter	
i.tasscap4	Tasseled Cap (Kauth Thomas) transformation for LANDSAT-TM 4 data
i.tasscap5	Tasseled Cap (Kauth Thomas) transformation for LANDSAT-TM 5 data
i.tasscap7	Tasseled Cap (Kauth Thomas) transformation for LANDSAT-TM 7 data
i.fft	Fast fourier transform (FFT) for image processing
i.ifft	Inverse fast fourier transform for image processing
r.describe	Prints terse list of category values found in a raster map layer
r.bitpattern	Compares bit patterns with a raster map
r.kappa	Calculate error matrix and kappa parameter for accuracy assessment of classification result
i.oif	Calculates optimal index factor table for landsat tm bands

B.7 GRASS Toolbox database modules

This Section lists all graphical dialogs in the GRASS Toolbox to manage, connect and work with internal and external databases. Working with spatial external databases is enabled via OGR and not covered by these modules.

Table 27: GRASS Toolbox: Database modules

Database management and analysis modules in the GRASS Toolbox	
Module name	Purpose
db.connect	Sets general DB connection mapset
db.connect.schema	Sets general DB connection mapset with a schema
v.db.reconnect.all	Reconnect vector to a new database
db.login	Set user/password for driver/database
db.in.ogr	Imports attribute tables in various formats
v.db.addtable	Create and add a new table to a vector
v.db.addcol	Adds one or more columns to the attribute table connected to a given vector map
v.db.dropcol	Drops a column from the attribute table connected to a given vector map
v.db.renamecol	Renames a column in a attribute table connected to a given vector map
v.db.update_const	Allows to assign a new constant value to a column
v.db.update_query	Allows to assign a new constant value to a column only if the result of a query is TRUE
v.db.update_op	Allows to assign a new value, result of operation on column(s), to a column in the attribute table connected to a given map
v.db.update_op_query	Allows to assign a new value to a column, result of operation on column(s), only if the result of a query is TRUE
db.execute	Execute any SQL statement
db.select	Prints results of selection from database based on SQL
v.db.select	Prints vector map attributes
v.db.select.where	Prints vector map attributes with SQL
v.db.join	Allows to join a table to a vector map table
v.db.univar	Calculates univariate statistics on selected table column for a GRASS vector map

B.8 GRASS Toolbox 3D modules

This Section lists all graphical dialogs in the GRASS Toolbox to work with 3D data. GRASS provides more modules, but they are currently only available using the GRASS Shell.

Table 28: GRASS Toolbox: 3D Visualization

3D visualization and analysis modules in the GRASS Toolbox	
Module name	Purpose
nviz	Open 3D-View in nviz

B.9 GRASS Toolbox help modules

The GRASS GIS Reference Manual offers a complete overview of the available GRASS modules, not limited to the modules and their often reduced functionalities implemented in the GRASS Toolbox.

Table 29: GRASS Toolbox: Reference Manual

Reference Manual modules in the GRASS Toolbox	
Module name	Purpose
g.manual	Display the HTML manual pages of GRASS

C Installation Guide

The following chapters provide build and installation information for QGIS Version 1.0.0. This document corresponds almost to a \LaTeX conversion of the INSTALL.t2t file coming with the QGIS sources from December, 16th 2008.

A current version is also available at the wiki, see: <http://wiki.qgis.org/qgiswiki/BuildingFromSource>

C.1 General Build Notes

At version 0.8.1 QGIS no longer uses the autotools for building. QGIS, like a number of major projects (eg. KDE 4.0), now uses cmake (<http://www.cmake.org>) for building from source. The configure script in this directory simply checks for the existence of cmake and provides some clues to build QGIS.

For complete information, see the wiki at: http://wiki.qgis.org/qgiswiki/Building_with_CMake

C.2 An overview of the dependencies required for building

Required build deps:

- CMake \geq 2.4.3
- Flex, Bison

Required runtime deps:

- Qt \geq 4.3.0
- Proj \geq ? (known to work with 4.4.x)
- GEOS \geq 2.2 (3.0 is supported, maybe 2.1.x works too)
- Sqlite3 \geq ? (probably 3.0.0)
- GDAL/OGR \geq 1.4.x

Optional dependencies:

- for GRASS plugin - GRASS \geq 6.0.0
- for georeferencer - GSL \geq ? (works with 1.8)
- for postgis support and SPIT plugin - PostgreSQL \geq 8.0.x
- for gps plugin - expat \geq ? (1.95 is OK)
- for mapserver export and PyQGIS - Python \geq 2.3 (2.5+ preferred)
- for PyQGIS - SIP \geq 4.5, PyQt \geq must match Qt version

Recommended runtime deps:

- for gps plugin - gpsbabel

D Building under windows using msys

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...

D.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.

D.2 Qt4.3

Download qt4.3 opensource precompiled edition exe and install (including the download and install of mingw) from here:

<http://www.trolltech.com/developer/downloads/qt/windows>

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\4.3.0\bin\qtvars.bat and add the following lines:

```
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\4.3.0\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\4.3.0\bin\qtvars.bat compile_debug`

Note: there is a problem when compiling debug version of Qt 4.3, 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:

```
c:\Qt\4.3.0 make
```

D.3 Flex and Bison

Note: I think this section can be removed as it should be installed into the msys image already.

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`

D.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.

D.4.1 Download and install Python - use Windows installer

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

<http://python.org/download/>

D.4.2 Download SIP and PyQt4 sources

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

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

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

D.4.3 Compile SIP

```
c:\Qt\4.3.0\bin\qtvars.bat
python configure.py -p win32-g++
make
make install
```

D.4.4 Compile PyQt

```
c:\Qt\4.3.0\bin\qtvars.bat
python configure.py
make
make install
```

D.4.5 Final python notes

Note: You can delete the directories with unpacked SIP and PyQt4 sources after a successful install, they're not needed anymore.

D.5 Subversion:

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

<http://subversion.tigris.org/files/documents/15/36797/svn-1.4.3-setup.exe>

D.6 CMake:

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

<http://www.cmake.org/files/v2.4/cmake-2.4.6-win32-x86.exe>

D.7 QGIS:

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

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

D BUILDING UNDER WINDOWS USING MSYS

Check out sources from SVN For svn head:

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

For svn 0.8 branch

```
svn co https://svn.osgeo.org/qgis/branches/Release-0_8_0 qgis0.8
```

D.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:

```
c:\Qt\4.3.0\bin\qtvars.bat
```

For ease of use add c:\Qt\4.3.0\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:

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

D.9 Configuration

```
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 cripts I recommend to leave the install prefix to its default c:\program files\

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

D.10 Compilation and installation

```
make make install
```

D.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.

The best way to do this is to download both the QGIS current release installer package from <http://qgis.org/uploadfiles/testbuilds/> and install it. Now copy the installation dir from C:\Program Files\Quantum GIS into c:\Program Files\qgis-0.8.1 (or whatever the current version is. The name should strictly match the version no.) After making this copy you can uninstall the release version of QGIS from your c:\Program Files directory using the provided uninstaller. Double check that the Quantum GIS dir is completely gone under program files afterwards.

Another 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.

D.12 Create the installation package: (optional)

Downlad 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'.

E Building on Mac OSX using frameworks and cmake (QGIS > 0.8)

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

Included are a few notes for building on Mac OS X 10.5 (Leopard).

E.1 Install XCODE

I recommend to get the latest xcode dmg from the Apple XDC Web site. Install XCODE after the ~941mb download is complete.

Note: It may be that you need to create some symlinks after installing the XCODE SDK (in particular if you are using XCODE 2.5 on tiger):

```
cd /Developer/SDKs/MacOSX10.4u.sdk/usr/  
sudo mv local/ local_  
sudo ln -s /usr/local/ local
```

E.2 Install Qt4 from .dmg

You need a minimum of Qt4.3.0. I suggest getting the latest (at time of writing).

```
ftp://ftp.trolltech.com/qt/source/qt-mac-opensource-4.3.2.dmg
```

If you want debug libs, Qt also provide a dmg with these:

```
ftp://ftp.trolltech.com/qt/source/qt-mac-opensource-4.3.2-debug-libs.dmg
```

I am going to proceed using only release libs at this stage as the download for the debug dmg is substantially bigger. If you plan to do any debugging though you probably want to get the debug libs dmg. Once downloaded open the dmg and run the installer.

Note: you need admin access to install.

After installing you need to make two small changes:

First edit `/Library/Frameworks/QtCore.framework/Headers/qconfig.h` and change

Note: this doesnt seem to be needed since version 4.2.3

```
QT_EDITION_Unknown to QT_EDITION_OPENSOURCE
```

Second change the default mkspec symlink so that it points to macx-g++:

```
cd /usr/local/Qt4.3/mkspecs/  
sudo rm default  
sudo ln -sf macx-g++ default
```

E.3 Install development frameworks for QGIS dependencies

Download William Kyngesburye's excellent all in one framework that includes proj, gdal, sqlite3 etc

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

Once downloaded, open and install the frameworks.

William provides an additional installer package for Postgresql/PostGIS. Its available here:

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

There are some additional dependencies that at the time of writing are not provided as frameworks so we will need to build these from source.

E.3.1 Additional Dependencies : GSL

Retrieve the Gnu Scientific Library from

```
curl -O ftp://ftp.gnu.org/gnu/gsl/gsl-1.8.tar.gz
```

Then extract it and build it to a prefix of /usr/local:

```
tar xvfz gsl-1.8.tar.gz
cd gsl-1.8
./configure --prefix=/usr/local
make
sudo make install
cd ..
```

E.3.2 Additional Dependencies : Expat

Get the expat sources:

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

```
tar xvfz expat-2.0.0.tar.gz
cd expat-2.0.0
./configure --prefix=/usr/local
make
sudo make install
cd ..
```

E.3.3 Additional Dependencies : SIP

Make sure you have the latest Python fom

<http://www.python.org/download/mac/>

Leopard note: Leopard includes a usable Python 2.5. Though you can install Python from python.org if preferred.

Retrieve the python bindings toolkit SIP from

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

Then extract and build it (this installs by default into the Python framework):

```
tar xvfz sip-<version number>.tar.gz
cd sip-<version number>
python configure.py
make
sudo make install
cd ..
```

Leopard notes

If building on Leopard, using Leopard's bundled Python, SIP wants to install in the system path – this is not a good idea. Use this configure command instead of the basic configure above:

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

E.3.4 Additional Dependencies : PyQt

If you encounter problems compiling PyQt using the instructions below you can also try adding python from your frameworks dir explicitly to your path e.g.

```
export PATH=/Library/Frameworks/Python.framework/Versions/Current/bin:$PATH$
```

Retrieve the python bindings toolkit for Qt from

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

Then extract and build it (this installs by default into the Python framework):

```
tar xvfz PyQt-mac<version number here>
cd PyQt-mac<version number here>
export QTDIR=/Developer/Applications/Qt
python configure.py
yes
make
sudo make install
cd ..
```

Leopard notes

If building on Leopard, using Leopard's bundled Python, PyQt wants to install in the system path – this is not a good idea. Use this configure command instead of the basic configure above:

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

There may be a problem with undefined symbols in QtOpenGL on Leopard. Edit QtOpenGL/makefile and add `-undefined dynamic_lookup` to `LFLAGS`.

E.3.5 Additional Dependencies : Bison

Leopard note: Leopard includes Bison 2.3, so this step can be skipped on Leopard.

The version of bison available by default on Mac OSX is too old so you need to get a more recent one on your system. Download it from:

```
curl -O http://ftp.gnu.org/gnu/bison/bison-2.3.tar.gz
```

Now build and install it to a prefix of `/usr/local` :

```
tar xvfz bison-2.3.tar.gz
cd bison-2.3
./configure --prefix=/usr/local
make
sudo make install
cd ..
```

E.4 Install CMAKE for OSX

Get the latest release from here:

<http://www.cmake.org/HTML/Download.html>

At the time of writing the file I grabbed was:

```
curl -O http://www.cmake.org/files/v2.4/cmake-2.4.6-Darwin-universal.dmg
```

Once downloaded open the dmg and run the installer

E.5 Install subversion for OSX

Leopard note: Leopard includes SVN, so this step can be skipped on Leopard.

The <http://sourceforge.net/projects/macsvn/> project has a downloadable build of svn. If you are a GUI inclined person you may want to grab their gui client too. Get the command line client here:

```
curl -O http://ufpr.dl.sourceforge.net/sourceforge/macsvn/Subversion_1.4.2.zip
```

Once downloaded open the zip file and run the installer.

You also need to install BerkleyDB available from the same <http://sourceforge.net/projects/macsvn/>. At the time of writing the file was here:

```
curl -O http://ufpr.dl.sourceforge.net/sourceforge/macsvn/Berkeley_DB_4.5.20.zip
```

Once again unzip this and run the installer therein. Lastly we need to ensure that the svn command-line executable is in the path. Add the following line to the end of `/etc/bashrc` using `sudo`:

```
sudo vim /etc/bashrc
```

And add this line to the bottom before saving and quitting:

```
export PATH=/usr/local/bin:$PATH:/usr/local/pgsql/bin
```

/usr/local/bin needs to be first in the path so that the newer bison (that will be built from source further down) is found before the bison (which is very old) that is installed by MacOSX.

Now close and reopen your shell to get the updated vars.

E.6 Check out QGIS from SVN

Now we are going to check out the sources for QGIS. First we will create a directory for working in:

```
mkdir -p ~/dev/cpp cd ~/dev/cpp
```

Now we check out the sources:

Trunk:

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

For svn 0.8 branch

```
svn co https://svn.osgeo.org/qgis/branches/Release-0_8_0 qgis0.8
```

For svn 0.9 branch

```
svn co https://svn.qgis.org/qgis/branches/Release-0_9_0 qgis0.9
```

The first time you check out QGIS sources you will probably get a message like this:

```
Error validating server certificate for 'https://svn.qgis.org:443':
- The certificate is not issued by a trusted authority. Use the fingerprint to
  validate the certificate manually! Certificate information:
- Hostname: svn.qgis.org
- Valid: from Apr  1 00:30:47 2006 GMT until Mar 21 00:30:47 2008 GMT
- Issuer: Developer Team, Quantum GIS, Anchorage, Alaska, US
- Fingerprint: 2f:cd:f1:5a:c7:64:da:2b:d1:34:a5:20:c6:15:67:28:33:ea:7a:9b
  (R)ject, accept (t)emporarily or accept (p)ermanently?
```

I suggest you press 'p' to accept the key permanently.

E.7 Configure the build

CMake supports out of source build so we will create a 'build' dir for the build process. By convention I build my software into a dir called 'apps' in my home directory. 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}/apps directory ...

```
cd qgis
mkdir build
cd build
cmake -D CMAKE_INSTALL_PREFIX=${HOME}/apps/ -D CMAKE_BUILD_TYPE=Release ..
```

Leopard note: To find the custom install of SIP on Leopard, add ""- D SIP_BINARY_PATH=/usr/local/bin/sip"" to the cmake command above, before the .. at the end, ie:

```
cmake -D CMAKE_INSTALL_PREFIX=${HOME}/apps/ -D CMAKE_BUILD_TYPE=Release -
D SIP_BINARY_PATH=/usr/local/bin/sip ..
```

To use the application build of GRASS on OSX, you can optionally use the following cmake invocation (minimum GRASS 6.3 required, substitute the GRASS version as required):

```
cmake -D CMAKE_INSTALL_PREFIX=${HOME}/apps/ \
-D GRASS_INCLUDE_DIR=/Applications/GRASS-6.3.app/Contents/MacOS/
include \
-D GRASS_PREFIX=/Applications/GRASS-6.3.app/Contents/MacOS \
-D CMAKE_BUILD_TYPE=Release \
..
```

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):

```
cmake -D CMAKE_INSTALL_PREFIX=${HOME}/apps/ \
-D GRASS_INCLUDE_DIR=/user/local/grass-6.3.0/include \
-D GRASS_PREFIX=/user/local/grass-6.3.0 \
-D CMAKE_BUILD_TYPE=Release \
..
```


E.8 Building

Now we can start the build process:

```
make
```

If all built without errors you can then install it:

```
make install
```

F Building on GNU/Linux

F.1 Building QGIS with Qt4.x

Requires: Ubuntu Hardy / Debian derived distro

These notes are current for Ubuntu 7.10 - 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.

F.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:

```
sudo apt-get update
```

F.3 Install Qt4

```
sudo apt-get install libqt4-core libqt4-debug \  
libqt4-dev libqt4-gui libqt4-qt3support libqt4-sql lsb-qt4 qt4-designer \  
qt4-dev-tools qt4-doc qt4-qtconfig uim-qt gcc libapt-pkg-perl resolvconf
```

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:

```
/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:

```
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.

F.4 Install additional software dependencies required by QGIS

```
sudo apt-get install gdal-bin libgdal1-dev libgeos-dev proj \  
libgdal-doc libhdf4g-dev libhdf4g-run python-dev \  
libgsl0-dev g++ libjasper-dev libtiff4-dev subversion \  

```

```
libsqlite3-dev sqlite3 ccache make libpq-dev flex bison cmake txt2tags \  
python-qt4 python-qt4-dev python-sip4 sip4 python-sip4-dev
```

Note: Debian users should use libgdal-dev above rather

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.

If you do not have cmake installed already:

```
sudo apt-get install cmake
```

F.5 GRASS Specific Steps

Note: If you don't need to build with GRASS support, you can skip this section.

Now you can install grass from dapper:

```
sudo apt-get install grass libgrass-dev libgdal1-1.4.0-grass
```

!\
You may need to explicitly state your grass version e.g. libgdal1-1.3.2-grass

F.6 Setup ccache (Optional)

You should also setup ccache to speed up compile times:

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

F.7 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:

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

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

F.8 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 privileges for the QGIS source repository, or use the developer checkout if you have permissions to commit source code changes.

1. Anonymous Checkout

```
cd ${HOME}/dev/cpp
svn co https://svn.osgeo.org/qgis/trunk/qgis qgis
```

2. Developer Checkout

```
cd ${HOME}/dev/cpp
svn co --username <yourusername> https://svn.osgeo.org/qgis/trunk/qgis qgis
```

The first time you check out the source you will be prompted to accept the qgis.org certificate. Press 'p' to accept it permanently:

```
Error validating server certificate for 'https://svn.qgis.org:443':
- The certificate is not issued by a trusted authority. Use the
  fingerprint to validate the certificate manually! Certificate
  information:
- Hostname: svn.qgis.org
- Valid: from Apr  1 00:30:47 2006 GMT until Mar 21 00:30:47 2008 GMT
- Issuer: Developer Team, Quantum GIS, Anchorage, Alaska, US
- Fingerprint:
  2f:cd:f1:5a:c7:64:da:2b:d1:34:a5:20:c6:15:67:28:33:ea:7a:9b (R)eject,
  accept (t)emporarily or accept (p)ermanently?
```

F.9 Starting the compile

Note: The next section describes howto build debian packages

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:

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

Now we create a build directory and run ccmake:

```
cd qgis
mkdir build
cd build
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:

```
make
make install
```

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

F.10 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:

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

The QGIS packages will be created with:

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

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

G CREATION OF MSYS ENVIRONMENT FOR COMPILATION OF QUANTUM GIS

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

```
sudo dpkg -i \  
../qgis_1.0preview16_amd64.deb \  
../libqgis-gui1_1.0preview16_amd64.deb \  
../libqgis-core1_1.0preview16_amd64.deb \  
../qgis-plugin-grass_1.0preview16_amd64.deb \  
../python-qgis_1.0preview16_amd64.deb
```

F.11 Running QGIS

Now you can try to run QGIS:

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

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

G Creation of MSYS environment for compilation of Quantum GIS

G.1 Initial setup

G.1.1 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).

G.1.2 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.

G.1.3 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`

G.2 Installing dependencies

G.2.1 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`

G.2.2 GDAL level one

Since Quantum GIS needs GDAL with GRASS support, we need to compile GDAL from source - Paul Kelly's package doesn't include GRASS support in GDAL. The idea is following:

1. compile GDAL without GRASS
2. compile GRASS
3. compile GDAL with GRASS

So, start with downloading GDAL sources:

<http://download.osgeo.org/gdal/gdal141.zip>

Unpack it to some directory, preferably `c:\msys\local\src`.

Start MSYS console, go to `gdal-1.4.1` directory and run the commands below. You can put them all to a script, e.g. `build-gdal.sh` and run them at once. The recipe is taken from Paul Kelly's instructions - basically they just make sure that the library will be created as DLL and the utility programs will be dynamically linked to it...

```
CFLAGS="-O2 -s" CXXFLAGS="-O2 -s" LDFLAGS=-s ./configure --without-libtool \
--prefix=/usr/local --enable-shared --disable-static --with-libz=/usr/local \
--with-png=/usr/local
make
make install
rm /usr/local/lib/libgdal.a
g++ -s -shared -o ./libgdal.dll -L/usr/local/lib -lz -lpng ./frmts/o/*.o ./gcore/*.o \
./port/*.o ./alg/*.o ./ogr/ogrsf_frmts/o/*.o ./ogr/ogrgeometryfactory.o \
./ogr/ogrpoint.o ./ogr/ogrcurve.o ./ogr/ogrlinestring.o ./ogr/ogrlinearring.o \
./ogr/ogrpolygon.o ./ogr/ogrutils.o ./ogr/ogrgeometry.o ./ogr/ogrgeometrycollection.o \
./ogr/ogrmultipolygon.o ./ogr/ogrsurface.o ./ogr/ogrmultipoint.o \
./ogr/ogrmultilinestring.o ./ogr/ogr_api.o ./ogr/ogrfeature.o ./ogr/ogrfeaturedefn.o \
./ogr/ogrfeaturequery.o ./ogr/ogrfeaturestyle.o ./ogr/ogrfielddefn.o \
./ogr/ogrspatialreference.o ./ogr/ogr_srsnode.o ./ogr/ogr_srs_proj4.o \
./ogr/ogr_fromepsg.o ./ogr/ogrct.o ./ogr/ogr_opt.o ./ogr/ogr_srs_esri.o \
./ogr/ogr_srs_pci.o ./ogr/ogr_srs_usgs.o ./ogr/ogr_srs_dict.o ./ogr/ogr_srs_panorama.o \
./ogr/swq.o ./ogr/ogr_srs_validate.o ./ogr/ogr_srs_xml.o ./ogr/ograssemblepolygon.o \
./ogr/ogr2gmlgeometry.o ./ogr/gml2ogrgeometry.o
```



```

install libgdal.dll /usr/local/lib
cd ogr
g++ -s ogrinfo.o -o ogrinfo.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s ogr2ogr.o -o ogr2ogr.exe -lgdal -L/usr/local/lib -lpng -lz -lgdal
g++ -s ogrtindex.o -o ogrtindex.exe -lgdal -L/usr/local/lib -lpng -lz -lgdal
install ogrinfo.exe ogr2ogr.exe ogrtindex.exe /usr/local/bin
cd ../apps
g++ -s gdalinfo.o -o gdalinfo.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s gdal_translate.o -o gdal_translate.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s gdaladdo.o -o gdaladdo.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s gdalwarp.o -o gdalwarp.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s gdal_contour.o -o gdal_contour.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s gdaltindex.o -o gdaltindex.exe -L/usr/local/lib -lpng -lz -lgdal
g++ -s gdal_rasterize.o -o gdal_rasterize.exe -L/usr/local/lib -lpng -lz -lgdal
install gdalinfo.exe gdal_translate.exe gdaladdo.exe gdalwarp.exe gdal_contour.exe \
gdaltindex.exe gdal_rasterize.exe /usr/local/bin

```

Finally, manually edit `gdal-config` in `c:\msys\local\bin` to replace the static library reference with `-lgdal`:

```
CONFIG_LIBS="-L/usr/local/lib -lpng -lz -lgdal"
```

GDAL build procedure can be greatly simplified to use `libtool` with a `libtool` line patch: configure `gdal` as below: `./configure --with-ngpython --with-xerces=/local/ --with-jasper=/local/ --with-grass=/local/grass-6.3.cvs/ --with-pg=/local/pgsql/bin/pg_config.exe`

Then fix `libtool` with: `mv libtool libtool.orig cat libtool.orig | sed 's/max_cmd_len=8192/max_cmd_len=32768/g' > libtool`

`Libtool` on windows assumes a line length limit of 8192 for some reason and tries to page the linking and fails miserably. This is a work around.

`Make` and `make install` should be hassle free after this.

G.2.3 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:

```
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 --with-cxx --without-jpeg --without-tiff --with-postgres=yes \
--with-postgres-includes=/local/pgsql/include --with-pgsql-libs=/local/pgsql/lib \
--with-opengl=windows --with-fftw --with-freetype \
--with-freetype-includes=/mingw/include/freetype2 \
--without-x --without-tcltk \
--enable-x11=no --enable-shared=yes --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>

G.2.4 GDAL level two

At this stage, we'll use GDAL sources we've used before, only the compilation will be a bit different.

But first in order to be able to compile GDAL sources with current GRASS CVS, you need to patch them, here's what you need to change:

http://trac.osgeo.org/gdal/attachment/ticket/1587/plugin_patch_grass63.diff

(you can patch it by hand or use `patch.exe` in `c:\msys\bin`)

Now in MSYS console go to the GDAL sources directory and run the same commands as in level one, only with these differences:

1) when running `./configure` add this argument:

```
--with-grass=/usr/local/grass-6.3.cvs
```

2) when calling `g++` on line 5 (which creates `libgdal.dll`), add these arguments:

```
-L/usr/local/grass-6.3.cvs/lib -lgrass\_vect -lgrass\_dig2 -lgrass\_dgl -lgrass\_rtree \
-lgrass\_linkm -lgrass\_dbmiclient -lgrass\_dbmibase -lgrass\_I -lgrass\_gproj \
-lgrass\_vask -lgrass\_gmath -lgrass\_gis -lgrass\_datetime}
```

Then again, edit `gdal-config` and change line with `CONFIG_LIBS`

```
CONFIG_LIBS="-L/usr/local/lib -lpng -L/usr/local/grass-6.3.cvs/lib -lgrass_vect \  
-lgrass_dig2 -lgrass_dgl -lgrass_rtree -lgrass_linkm -lgrass_dbmiclient \  
-lgrass_dbmibase -lgrass_I -lgrass_gproj -lgrass_vask -lgrass_gmath -lgrass_gis \  
-lgrass_datetime -lz -L/usr/local/lib -lgdal"
```

Now, GDAL should be able to work also with GRASS raster layers.

G.2.5 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:

```
#ifdef _WIN32
```

to:

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

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

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

G.2.6 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

G.2.7 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:

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

G.2.8 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:

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

G.2.9 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%2Fpostgresql-8.2.4-1-binaries-no-installer.zip>

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

G.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.

H Building with MS Visual Studio

!\ This section describes a process where you build all dependencies yourself. See the section after this for a simpler procedure where we have all the dependencies you need pre-packaged and we focus just on getting Visual Studio Express set up and building QGIS.

Note: that this does not currently include GRASS or Python plugins.

H.1 Setup Visual Studio

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

H.1.1 Express Edition

The free Express Edition lacks the platform SDK which contains headers and so on that are needed when building QGIS. The platform SDK can be installed as described here:

<http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/>

Once this is done, you will need to edit the `<vsinstalldir>\Common7\Tools\vsvars` file as follows:

```
Add %PlatformSDKDir%\Include\atl and %PlatformSDKDir%\Include\mfc to the
@set INCLUDE entry.
```

This will add more headers to the system INCLUDE path. **Note:** that this will only work when you use the Visual Studio command prompt when building. Most of the dependencies will be built with this. You will also need to perform the edits described here to remove the need for a library that Visual Studio Express lacks:

<http://www.codeproject.com/wtl/WTLExpress.asp>

H.1.2 All Editions

You will need `stdint.h` and `unistd.h`. `unistd.h` comes with GnuWin32 version of flex & bison binaries (see later). `stdint.h` can be found here:

<http://www.azillionmonkeys.com/qed/pstdint.h>

Copy both of these to `<vsinstalldir>\VC\include`.

H.2 Download/Install Dependencies

This section describes the downloading and installation of the various QGIS dependencies.

H.2.1 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 and run the installers:

<http://gnuwin32.sourceforge.net/downlinks/flex.php>

<http://gnuwin32.sourceforge.net/downlinks/bison.php>

H.2.2 To include PostgreSQL support in Qt

If you want to build Qt with PostgreSQL support you need to download PostgreSQL, install it and create a library you can later link with Qt.

Download from `.../binary/v8.2.5/win32/postgresql-8.2.5-1.zip` from an PostgreSQL.org Mirror and install.

PostgreSQL is currently build with MinGW and comes with headers and libraries for MinGW. The headers can be used with Visual C++ out of the box, but the library is only shipped in DLL and archive (.a) form and therefore cannot be used with Visual C++ directly.

To create a library copy following sed script to the file `mkdef.sed` in PostgreSQL lib directory:

```
/Dump of file / {  
s/Dump of file \([^ ]*\)$ /LIBRARY \1/p  
a\  
EXPORTS
```

```

}
/[ ]*ordinal hint/,/^[ ]*Summary/ {
  /^[ ]*\+[0-9]\+/ {
    s/^[ ]*\+[0-9]\+[ ]*\+[0-9A-Fa-f]\+[ ]*\+[0-9A-Fa-f]\+[ ]*\+([^\s=]\+\.*)$/ \1/p
  }
}
}

```

and process execute in the Visual Studio C++ command line (from Programs menu):

```

cd c:\Program Files\PostgreSQL\8.2\bin
dumpbin /exports ..\bin\libpq.dll | sed -nf ../lib/mkdef.sed >..\lib\libpq.def
cd ..\lib
lib /def:libpq.def /machine:x86

```

You'll need an sed for that to work in your path (e.g. from cygwin or msys).

That's almost it. You only need to the include and lib path to INCLUDE and LIB in vcvars.bat respectively.

H.2.3 Qt

Build Qt following the instructions here:

http://wiki.qgis.org/qgiswiki/Building_QT_4_with_Visual_C%2B%2B_2005

H.2.4 Proj.4

Get proj.4 source from here:

<http://proj.maptools.org/>

Using the Visual Studio command prompt (ensures the environment is setup properly), run the following in the src directory:

```
nmake -f makefile.vc
```

Install by running the following in the top level directory setting PROJ_DIR as appropriate:

```
set PROJ_DIR=c:\lib\proj
```

```
mkdir %PROJ_DIR%\bin
mkdir %PROJ_DIR%\include
mkdir %PROJ_DIR%\lib

copy src\*.dll %PROJ_DIR%\bin
copy src\*.exe %PROJ_DIR%\bin
copy src\*.h %PROJ_DIR%\include
copy src\*.lib %PROJ_DIR%\lib
```

This can also be added to a batch file.

H.2.5 GSL

Get gsl source from here:

<http://david.geldreich.free.fr/downloads/gsl-1.9-windows-sources.zip>

Build using the gsl.sln file

H.2.6 GEOS

Get geos from svn (svn checkout <http://svn.refractor.net/geos/trunk> geos). Edit geos\source\makefile.vc as follows:

Uncomment lines 333 and 334 to allow the copying of version.h.vc to version.h.

Uncomment lines 338 and 339.

Rename geos_c.h.vc to geos_c.h.in on lines 338 and 339 to allow the copying of geos_c.h.in to geos_c.h.

Using the Visual Studio command prompt (ensures the environment is setup properly), run the following in the top level directory:

```
nmake -f makefile.vc
```

Run the following in top level directory, setting GEOS_DIR as appropriate:

```
set GEOS_DIR="c:\lib\geos"

mkdir %GEOS_DIR%\include
```



```
mkdir %GEOS_DIR%\lib
mkdir %GEOS_DIR%\bin

xcopy /S/Y source\headers\*.h %GEOS_DIR%\include
copy /Y capi\*.h %GEOS_DIR%\include
copy /Y source\*.lib %GEOS_DIR%\lib
copy /Y source\*.dll %GEOS_DIR%\bin
```

This can also be added to a batch file.

H.2.7 GDAL

Get gdal from svn (svn checkout <https://svn.osgeo.org/gdal/branches/1.4/gdal> gdal).

Edit nmake.opt to suit, it's pretty well commented.

Using the Visual Studio command prompt (ensures the environment is setup properly), run the following in the top level directory:

```
nmake -f makefile.vc
```

and

```
nmake -f makefile.vc devinstall
```

H.2.8 PostGIS

Get PostGIS and the Windows version of PostgreSQL from here:

<http://postgis.refractor.net/download/>

Note: the warning about not installing the version of PostGIS that comes with the PostgreSQL installer. Simply run the installers.

H.2.9 Expat

Get expat from here:

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

You'll need `expat-win32bin-2.0.1.exe`.

Simply run the executable to install `expat`.

H.2.10 CMake

Get CMake from here:

<http://www.cmake.org/HTML/Download.html>

You'll need `cmake-<version>-win32-x86.exe`. Simply run this to install CMake.

H.3 Building QGIS with CMAKE

Get QGIS source from svn (`svn co https://svn.osgeo.org/qgis/trunk/qgis qgis`).

Create a 'Build' directory in the top level QGIS directory. This will be where all the build output will be generated.

Run `Start->All Programs->CMake->CMake`.

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 in the top level QGIS directory.

Fill in the various `*_INCLUDE_DIR` and `*_LIBRARY` entries in the 'Cache Values' list.

Click the Configure button. You will be prompted for the type of makefile that will be generated. Select Visual Studio 8 2005 and click OK.

All being well, configuration should complete without errors. If there are errors, it is usually due to an incorrect path to a header or library directory. Failed items will be shown in red in the list.

Once configuration completes without error, click OK to generate the solution and project files.

With Visual Studio 2005, open the `qgis.sln` file that will have been created in the Build directory you created earlier.

Build the `ALL_BUILD` project. This will build all the QGIS binaries along with all the plugins.

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).

You will also either need to add all the dependency dlls to the QGIS install directory or add their

respective directories to your PATH.

I Building under Windows using MSVC Express

Note:: Building under MSVC is still a work in progress. In particular the following dont work yet: python, grass, postgis connections.

/!\ This section of the document is in draft form and is not ready to be used yet.

Tim Sutton, 2007

I.1 System preparation

I started with a clean XP install with Service Pack 2 and all patches applied. I have already compiled all the dependencies you need for gdal, expat etc, so this tutorial wont cover compiling those from source too. Since compiling these dependencies was a somewhat painful task I hope my pre-compiled libs will be adequate. If not I suggest you consult the individual projects for specific build documentation and support. Lets go over the process in a nutshell before we begin:

- * Install XP (I used a Parallels virtual machine)
- * Install the premade libraries archive I have made for you
- * Install Visual Studio Express 2005 sp1
- * Install the Microsoft Platform SDK
- * Install command line subversion client
- * Install library dependencies bundle
- * Install Qt 4.3.2
- * Check out QGIS sources
- * Compile QGIS
- * Create setup.exe installer for QGIS

I.2 Install the libraries archive

Half of the point of this section of the MSVC setup procedure is to make things as simple as possible for you. To that end I have prepared an archive that includes all dependencies needed to build QGIS except Qt (which we will build further down). Fetch the archive from:

http://qgis.org/uploadfiles/msvc/qgis_msvc_deps_except_qt4.zip

Create the following directory structure:

```
c:\dev\cpp\
```

And then extract the libraries archive into a subdirectory of the above directory so that you end up with:

c:\dev\cpp\qgislibs-release

Note: that you are not obliged to use this directory layout, but you should adjust any instructions that follow if you plan to do things differently.

I.3 Install Visual Studio Express 2005

First thing we need to get is MSVC Express from here:

<http://msdn2.microsoft.com/en-us/express/aa975050.aspx>

The page is really confusing so dont feel bad if you cant actually find the download at first! There are six coloured blocks on the page for the various studio family members (vb / c# / j# etc). Simply choose your language under the 'select your language' combo under the yellow C++ block, and your download will begin. Under internet explorer I had to disable popup blocking for the download to be able to commence.

Once the setup commences you will be prompted with various options. Here is what I chose :

* Send usage information to Microsoft (No) * Install options: * Graphical IDE (Yes) * Microsoft MSDN Express Edition (No) * Microsoft SQL Server Express Edition (No) * Install to folder: C:\Program Files\Microsoft Visual Studio 8\ (default)

It will need to download around 90mb of installation files and reports that the install will consume 554mb of disk space.

I.4 Install Microsoft Platform SDK2

Go to this page:

<http://msdn2.microsoft.com/en-us/express/aa700755.aspx>

Start by using the link provided on the above page to download and install the platform SDK2.

The actual SDK download page is once again a bit confusing since the links for downloading are hidden amongst a bunch of other links. Basically look for these three links with their associated 'Download' buttons and choose the correct link for your platform:

PSDK-amd64.exe	1.2 MB	Download
PSDK-ia64.exe	1.3 MB	Download
PSDK-x86.exe	1.2 MB	Download

When you install make sure to choose 'custom install'. These instructions assume you are installing into the default path of:

C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\

We will go for the minimal install that will give us a working environment, so on the custom installation screen I made the following choices:

Configuration Options

+ Register Environmental Variables	(Yes)
Microsoft Windows Core SDK	
+ Tools	(Yes)
+ Tools (AMD 64 Bit)	(No unless this applies)
+ Tools (Intel 64 Bit)	(No unless this applies)
+ Build Environment	
+ Build Environment (AMD 64 Bit)	(No unless this applies)
+ Build Environment (Intel 64 Bit)	(No unless this applies)
+ Build Environment (x86 32 Bit)	(Yes)
+ Documentation	(No)
+ Redistributable Components	(Yes)
+ Sample Code	(No)
+ Source Code	(No)
+ AMD 64 Source	(No)
+ Intel 64 Source	(No)
Microsoft Web Workshop	(Yes) (needed for shlwapi.h)
+ Build Environment	(Yes)
+ Documentation	(No)
+ Sample Code	(No)
+ Tools	(No)
Microsoft Internet Information Server (IIS) SDK	(No)
Microsoft Data Access Services (MDAC) SDK	(Yes) (needed by GDAL for odbc)
+ Tools	
+ Tools (AMD 64 Bit)	(No)
+ Tools (AMD 64 Bit)	(No)
+ Tools (x86 32 Bit)	(Yes)
+ Build Environment	
+ Tools (AMD 64 Bit)	(No)
+ Tools (AMD 64 Bit)	(No)
+ Tools (x86 32 Bit)	(Yes)
+ Documentation	(No)
+ Sample Code	(No)

I BUILDING UNDER WINDOWS USING MSVC EXPRESS

Microsoft Installer SDK	(No)
Microsoft Table PC SDK	(No)
Microsoft Windows Management Instrumentation	(No)
Microsoft DirectShow SDK	(No)
Microsoft Media Services SDK	(No)
Debuggin Tools for Windows	(Yes)

Note: that you can always come back later to add extra bits if you like.

Note: that installing the SDK requires validation with the Microsoft Genuine Advantage application. Some people have a philosophical objection to installing this software on their computers. If you are one of them you should probably consider using the MINGW build instructions described elsewhere in this document.

The SDK installs a directory called

```
C:\Office10
```

Which you can safely remove.

After the SDK is installed, follow the remaining notes on the page link above to get your MSVC Express environment configured correctly. For your convenience, these are summarised again below, and I have added a couple more paths that I discovered were needed:

- 1) open Visual Studio Express IDE
- 2) Tools -> Options -> Projects and Solutions -> VC++ Directories
- 3) Add:

Executable files:

```
C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Bin
```

Include files:

```
C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Include
```

```
C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Include\atl
```

```
C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Include\mfc
```

Library files: C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Lib

- 4) Close MSVC Express IDE
- 5) Open the following file with notepad:

```
C:\Program Files\Microsoft Visual Studio 8\VC\VCProjectDefaults\corewin_express.vsprops
```

and change the property:

```
AdditionalDependencies="kernel32.lib"
```

To read:

```
AdditionalDependencies="kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib  
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib"
```

The notes go on to show how to build a mswin32 application which you can try if you like - I'm not going to recover that here.

I.5 Edit your vsvars

Backup your vsvars32.bat file in

```
C:\Program Files\Microsoft Visual Studio 8\Common7\Tools
```

and replace it with this one:

```
@SET VSINSTALLDIR=C:\Program Files\Microsoft Visual Studio 8  
@SET VCINSTALLDIR=C:\Program Files\Microsoft Visual Studio 8\VC  
@SET FrameworkDir=C:\WINDOWS\Microsoft.NET\Framework  
@SET FrameworkVersion=v2.0.50727  
@SET FrameworkSDKDir=C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0  
@if "%VSINSTALLDIR%"==" " goto error_no_VSINSTALLDIR  
@if "%VCINSTALLDIR%"==" " goto error_no_VCINSTALLDIR  
  
@echo Setting environment for using Microsoft Visual Studio 2005 x86 tools.  
  
@rem  
@rem Root of Visual Studio IDE installed files.  
@rem  
@set DevEnvDir=C:\Program Files\Microsoft Visual Studio 8\Common7\IDE  
  
@set PATH=C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;C:\Program \  
Files\Microsoft Visual Studio 8\VC\BIN;C:\Program Files\Microsoft Visual Studio 8\ \  
Common7\Tools;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\bin; \  
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual \
```

```
Studio 8\VC\VCPackages;%PATH%
@rem added by Tim
@set PATH=C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Bin;%PATH%
@set INCLUDE=C:\Program Files\Microsoft Visual Studio 8\VC\INCLUDE; \
%INCLUDE%
@rem added by Tim
@set INCLUDE=C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\ \
Include;%INCLUDE%
@set INCLUDE=C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\ \
Include\mfc;%INCLUDE%
@set INCLUDE=%INCLUDE%;C:\dev\cpp\qgislibs-release\include\postgresql
@set LIB=C:\Program Files\Microsoft Visual Studio 8\ \
VC\LIB;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\lib;%LIB%
@rem added by Tim
@set LIB=C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Lib;%LIB%
@set LIB=%LIB%;C:\dev\cpp\qgislibs-release\lib
@set LIBPATH=C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727

@goto end

:error_no_VSINSTALLDIR
@echo ERROR: VSINSTALLDIR variable is not set.
@goto end

:error_no_VCINSTALLDIR
@echo ERROR: VCINSTALLDIR variable is not set.
@goto end

:end
```

I.6 Environment Variables

Right click on 'My computer' then select the 'Advanced' tab. Click environment variables and create or augment the following "System" variables (if they dont already exist):

Variable Name:	Value:

EDITOR	vim
INCLUDE	C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2 \
	\Include\.

```

LIB          C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2 \
\lib\.
LIB_DIR      C:\dev\cpp\qgislibs-release
PATH         C:\Program Files\CMake 2.4\bin;
            %SystemRoot%\system32;
            %SystemRoot%;
            %SystemRoot%\System32\Wbem;
            C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2 \
            \Bin\.;
            C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\ \
            \Bin\WinNT\;
            C:\Program Files\svn\bin;C:\Program Files\Microsoft Visual Studio 8 \
            \VC\bin;
            C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;
            "c:\Program Files\Microsoft Visual Studio 8\Common7\Tools";
            c:\Qt\4.3.2\bin;
            "C:\Program Files\PuTTY"
QTDIR       c:\Qt\4.3.2
SVN_SSH     "C:\\Program Files\\PuTTY\\plink.exe"

```

I.7 Building Qt4.3.2

You need a minimum of Qt 4.3.2 here since this is the first version to officially support building the open source version of Qt for windows under MSVC.

Download Qt 4.x.x source for windows from

<http://www.trolltech.com>

Unpack the source to

c:\Qt\4.x.x\

I.7.1 Compile Qt

Open the Visual Studio C++ command line and cd to c:\Qt\4.x.x where you extracted the source and enter:

```
configure -platform win32-msvc2005
```

```
nmake  
nmake install
```

Add `-qt-sql-odbc -qt-sql-psql` to the configure line if you want odbc and PostgreSQL support build into Qt.

Note: For me in some cases I got a build error on `qsscreenshot.pro`. If you are only interested in having the libraries needed for building Qt apps, you can probably ignore that. Just check in `c:\Qt\4.3.2\bin` to check all dlls and helper apps (assistant etc) have been made.

I.7.2 Configure Visual C++ to use Qt

After building configure the Visual Studio Express IDE to use Qt:

- 1) open Visual Studio Express IDE
- 2) Tools -> Options -> Projects and Solutions -> VC++ Directories
- 3) Add:

Executable files:

```
$(QTDIR)\bin
```

Include files:

```
$(QTDIR)\include  
$(QTDIR)\include\Qt  
$(QTDIR)\include\QtCore  
$(QTDIR)\include\QtGui  
$(QTDIR)\include\QtNetwork  
$(QTDIR)\include\QtSvg  
$(QTDIR)\include\QtXml  
$(QTDIR)\include\Qt3Support  
$(LIB_DIR)\include (needed during qgis compile to find stdint.h and unistd.h)
```

Library files:

```
$(QTDIR)\lib
```

Source Files:

```
$(QTDIR)\src
```

Hint: You can also add

```
QString = t=<d->data, su>, size=<d->size, i>
```

to AutoExp.DAT in C:\Program Files\Microsoft Visual Studio 8\Common7\Packages\Debugger before

```
[Visualizer]
```

That way the Debugger will show the contents of QString when you point at or watch a variable in the debugger. There are probably much more additions - feel free to add some - I just needed QString and took the first hit in google I could find.

I.8 Install Python

Download <http://python.org/ftp/python/2.5.1/python-2.5.1.msi> and install it.

I.9 Install SIP

Download <http://www.riverbankcomputing.com/Downloads/sip4/sip-4.7.1.zip> and extract it into your c:\dev\cpp directory. From a Visual C++ command line cd to the directory where you extract SIP and run:

```
c:\python25\python configure.py -p win32-msvc2005
nmake
nmake install
```

I.10 Install PyQt4

Download <http://www.riverbankcomputing.com/Downloads/PyQt4/GPL/PyQt-win-gpl-4.3.1.zip> and extract it into your c:\dev\cpp directory. From a Visual C++ command line cd to the directory where you extracted PyQt4 and run:

```
c:\python25\python configure.py -p win32-msvc2005
nmake
nmake install
```

I.11 Install CMake

Download and install cmake 2.4.7 or better, making sure to enable the option: Update path for all users

I.12 Install Subversion

You "must" install the command line version if you want the CMake svn scripts to work. Its a bit tricky to find the correct version on the subversion download site as they have som misleadingly named similar downloads. Easiest is to just get this file:

<http://subversion.tigris.org/downloads/1.4.5-win32/apache-2.2/svn-win32-1.4.5.zip>

Extract the zip file to

```
C:\Program Files\svn
```

And then add

```
C:\Program Files\svn\bin
```

To your path.

I.13 Initial SVN Check out

Open a cmd.exe window and do:

```
cd \  
cd dev  
cd cpp  
svn co https://svn.osgeo.org/qgis/trunk/qgis
```

At this point you will probably get a message like this:

```
C:\dev\cpp>svn co https://svn.osgeo.org/qgis/trunk/qgis  
Error validating server certificate for 'https://svn.qgis.org:443':  
- The certificate is not issued by a trusted authority. Use the  
  fingerprint to validate the certificate manually!  
Certificate information:
```

```
- Hostname: svn.qgis.org
- Valid: from Sat, 01 Apr 2006 03:30:47 GMT until Fri, 21 Mar 2008 03:30:47 GMT
- Issuer: Developer Team, Quantum GIS, Anchorage, Alaska, US
- Fingerprint: 2f:cd:f1:5a:c7:64:da:2b:d1:34:a5:20:c6:15:67:28:33:ea:7a:9b
(R)eject, accept (t)emporarily or accept (p)ermanently?
```

Press 'p' to accept and the svn checkout will commence.

I.14 Create Makefiles using cmake.exe

I won't be giving a detailed description of the build process, because the process is explained in the first section (where you manually build all dependencies) of the windows build notes in this document. Just skip past the parts where you need to build GDAL etc, since this simplified install process does all the dependency provisioning for you.

```
cd qgis
mkdir build
cd build
cmake ..
```

Cmake should find all dependencies for you automatically (it uses the LIB_DIR environment to find them all in c:\dev\cpp\qgislibs-release). Press configure again after the cmake gui appears and when all the red fields are gone, and you have made any personalisations to the setup, press ok to close the cmake gui.

Now open Visual Studio Express and do: File -> Open -> Project / Solution

Now open the cmake generated QGIS solution which should be in :

```
c:\dev\cpp\qgis\build\qgisX.X.X.sln
```

Where X.X.X represents the current version number of QGIS. Currently I have only made release built dependencies for QGIS (debug versions will follow in future), so you need to be sure to select 'Release' from the solution configurations toolbar. Next right click on ALL_BUILD in the solution browser, and then choose build. Once the build completes right click on INSTALL in the solution browser and choose build. This will by default install qgis into c:\program files\qgisX.X.X.

I.15 Running and packaging

To run QGIS you need to at the minimum copy the dlls from c:\dev\cpp\qgislibs-release\bin into the c:\program files\qgisX.X.X directory.

J QGIS Coding Standards

The following chapters provide coding information for QGIS Version 1.0.0. This document corresponds almost to a \LaTeX conversion of the CODING.t2t file coming with the QGIS sources from December, 16th 2008.

These standards should be followed by all QGIS developers. Current information about QGIS Coding Standards are also available from wiki at:

<http://wiki.qgis.org/qgiswiki/CodingGuidelines>
<http://wiki.qgis.org/qgiswiki/CodingStandards>
<http://wiki.qgis.org/qgiswiki/UsingSubversion>
<http://wiki.qgis.org/qgiswiki/DebuggingPlugins>
<http://wiki.qgis.org/qgiswiki/DevelopmentInBranches>
<http://wiki.qgis.org/qgiswiki/SubmittingPatchesAndSvnAccess>

J.1 Classes

J.1.1 Names

Class in QGIS begin with Qgs and are formed using mixed case.

Examples:

QgsPoint
QgsMapCanvas
QgsRasterLayer

J.1.2 Members

Class member names begin with a lower case *m* and are formed using mixed case.

mMapCanvas
mCurrentExtent

All class members should be private. **Public class members are STRONGLY discouraged**

J.1.3 Accessor Functions

Class member values should be obtained through accessor functions. The function should be named without a *get* prefix. Accessor functions for the two private members above would be:

```
mapCanvas()  
currentExtent()
```

J.1.4 Functions

Function names begin with a lowercase letter and are formed using mixed case. The function name should convey something about the purpose of the function.

```
updateMapExtent()  
setUserOptions()
```

J.2 Qt Designer

J.2.1 Generated Classes

QGIS classes that are generated from Qt Designer (ui) files should have a *Base* suffix. This identifies the class as a generated base class.

Examples:

```
QgsPluginMangerBase  
QgsUserOptionsBase
```

J.2.2 Dialogs

All dialogs should implement the following: * Tooltip help for all toolbar icons and other relevant widgets * WhatsThis help for **all** widgets on the dialog * An optional (though highly recommended) context sensitive *Help* button that directs the user to the appropriate help page by launching their web browser

J.3 C++ Files

J.3.1 Names

C++ implementation and header files should have a .cpp and .h extension respectively. Filename should be all lowercase and, in the case of classes, match the class name.

Example:

Class QgsFeatureAttribute source files are
qgsfeatureattribute.cpp and qgsfeatureattribute.h

J.3.2 Standard Header and License

Each source file should contain a header section patterned after the following example:

```
/*
 *
 * qgsfield.cpp - Describes a field in a layer or table
 * -----
 *
 * Date                : 01-Jan-2004
 * Copyright           : (C) 2004 by Gary E.Sherman
 * Email              : sherman at mrcc.com
 *
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 *
 *
 */
```

J.3.3 CVS Keyword

Each source file should contain the \$Id\$ keyword. This will be expanded by CVS to contain useful information about the file, revision, last committer, and date/time of last checkin.

Place the keyword right after the standard header/license that is found at the top of each source file:

```
/* $Id$ */
```


J.4 Variable Names

Variable names begin with a lower case letter and are formed using mixed case.

Examples:

```
mapCanvas
```

```
currentExtent
```

J.5 Enumerated Types

Enumerated types should be named in CamelCase with a leading capital e.g.:

```
enum UnitType
{
    Meters,
    Feet,
    Degrees,
    UnknownUnit
} ;
```

Do not use generic type names that will conflict with other types. e.g. use "UnkownUnit" rather than "Unknown"

J.6 Global Constants

Global constants should be written in upper case underscore separated e.g.:

```
const long GEOCRS_ID = 3344;
```

J.7 Editing

Any text editor/IDE can be used to edit QGIS code, providing the following requirements are met.

J.7.1 Tabs

Set your editor to emulate tabs with spaces. Tab spacing should be set to 2 spaces.

J.7.2 Indentation

Source code should be indented to improve readability. There is a `.indent.pro` file in the QGIS `src` directory that contains the switches to be used when indenting code using the GNU indent program. If you don't use GNU indent, you should emulate these settings.

J.7.3 Braces

Braces should start on the line following the expression:

```
if(foo == 1)
{
    // do stuff
    ...
}else
{
    // do something else
    ...
}
```

J.8 API Compatibility

From QGIS 1.0 we will provide a stable, backwards compatible API. This will provide a stable basis for people to develop against, knowing their code will work against any of the 1.x QGIS releases (although recompiling may be required). Cleanups to the API should be done in a manner similar to the Trolltech developers e.g.

```
class Foo
{
    public:
        /** This method will be deprecated, you are encouraged to use
            doSomethingBetter() rather.
            @see doSomethingBetter()
            */
        bool doSomething();

        /** Does something a better way.
            @note This method was introduced in QGIS version 1.1
            */
}
```

```
bool doSomethingBetter();  
  
}
```

J.9 Coding Style

Here are described some programming hints and tips that will hopefully reduce errors, development time, and maintenance.

J.9.1 Where-ever Possible Generalize Code

If you are cut-n-pasting code, or otherwise writing the same thing more than once, consider consolidating the code into a single function.

This will allow changes to be made in one location instead of in multiple places

- help prevent code bloat
- make it more difficult for multiple copies to evolve differences over time, thus making it harder to understand and maintain for others

J.9.2 Prefer Having Constants First in Predicates

Prefer to put constants first in predicates.

```
"0 == value" instead of "value == 0"
```

This will help prevent programmers from accidentally using "=" when they meant to use "==", which can introduce very subtle logic bugs. The compiler will generate an error if you accidentally use "=" instead of==" for comparisons since constants inherently cannot be assigned values.

J.9.3 Whitespace Can Be Your Friend

Adding spaces between operators, statements, and functions makes it easier for humans to parse code.

Which is easier to read, this:

```
if (!a&&b)
```

or this:

```
if ( ! a && b )
```

J.9.4 Add Trailing Identifying Comments

Adding comments at the end of function, struct and class implementations makes it easier to find them later.

Consider that you're at the bottom of a source file and need to find a very long function – without these kinds of trailing comments you will have to page up past the body of the function to find its name. Of course this is ok if you wanted to find the beginning of the function; but what if you were interested at code near its end? You'd have to page up and then back down again to the desired part.

E.g.,

```
void foo::bar()
{
    // ... imagine a lot of code here
} // foo::bar()
```

J.9.5 Use Braces Even for Single Line Statements

Using braces for code in if/then blocks or similar code structures even for single line statements means that adding another statement is less likely to generate broken code.

Consider:

```
if (foo)
    bar();
else
    baz();
```

Adding code after bar() or baz() without adding enclosing braces would create broken code. Though most programmers would naturally do that, some may forget to do so in haste.

So, prefer this:

```
if (foo)
{
```

```
    bar();
}
else
{
    baz();
}
```

J.9.6 Book recommendations

Effective C++ <http://www.awprofessional.com/title/0321334876>

More Effective C++ <http://www.awprofessional.com/bookstore/product.asp?isbn=020163371X&rl=1>

Effective STL <http://www.awprofessional.com/title/0201749629>

Design Patterns <http://www.awprofessional.com/title/0201634988>

You should also really read this article from Qt Quarterly on designing Qt style <http://doc.trolltech.com/qq/qq13-apis.html>

K SVN Access

This page describes how to get started using the QGIS Subversion repository

K.1 Accessing the Repository

To check out QGIS HEAD:

```
svn --username [your user name] co https://svn.qgis.org/repos/qgis/trunk/qgis
```

K.2 Anonymous Access

You can use the following commands to perform an anonymous checkout from the QGIS Subversion repository. Note we recommend checking out the trunk (unless you are a developer or really HAVE to have the latest changes and dont mind lots of crashing!).

You must have a subversion client installed prior to checking out the code. See the Subversion website for more information. The Links page contains a good selection of SVN clients for various platforms.

To check out a branch

```
svn co https://svn.qgis.org/repos/qgis/branches/<branch name>
```

To check out SVN stable trunk:

```
svn co https://svn.qgis.org/repos/qgis/trunk/qgis qgis_unstable
```

!\ **Note:** If you are behind a proxy server, edit your `~/subversion/servers` file to specify your proxy settings first!

!\ **Note:** In QGIS we keep our most stable code in trunk. Periodically we will tag a release off trunk, and then continue stabilisation and selective incorporation of new features into trunk.

See the `INSTALL` file in the source tree for specific instructions on building development versions.

K.3 QGIS documentation sources

If you're interested in checking out Quantum GIS documentation sources:

```
svn co https://svn.qgis.org/repos/qgis_docs/trunk qgis_docs
```

You can also take a look at `DocumentationWritersCorner` for more information.

K.4 Documentation

The repository is organized as follows:

<http://wiki.qgis.org/images/repo.png>

See the Subversion book <http://svnbook.red-bean.com> for information on becoming a SVN master.

K.5 Development in branches

K.5.1 Purpose

The complexity of the QGIS source code has increased considerably during the last years. Therefore it is hard to anticipate the side effects that the addition of a feature will have. In the past, the QGIS project had very long release cycles because it was a lot of work to reestablish the stability of the software system after new features were added. To overcome these problems, QGIS switched to a development model where new features are coded in svn branches first and merged to trunk (the main branch) when they are finished and stable. This section describes the procedure for branching and merging in the QGIS project.

K.5.2 Procedure

* Initial announcement on mailing list Before starting, make an announcement on the developer mailing list to see if another developer is already working on the same feature. Also contact the technical advisor of the project steering committee (PSC). If the new feature requires any changes to the QGIS architecture, a request for comment (RFC) is needed. * Create a branch Create a new svn branch for the development of the new feature (see UsingSubversion for the svn syntax). Now you can start developing. * Merge from trunk regularly It is recommended to merge the changes in trunk to the branch on a regular basis. This makes it easier to merge the branch back to trunk later. * Documentation on wiki It is also recommended to document the intended changes and the current status of the work on a wiki page. * Testing before merging back to trunk When you are finished with the new feature and happy with the stability, make an announcement on the developer list. Before merging back, the changes will be tested by developers and users. Binary packages (especially for OSX and Windows) will be generated to also involve non-developers. In trac, a new Component will be opened to file tickets against. Once there are no remaining issues left, the technical advisor of the PSC merges the changes into trunk.

K.5.3 Creating a branch

We prefer that new feature developments happen out of trunk so that trunk remains in a stable state. To create a branch use the following command:

```
svn copy https://svn.qgis.org/repos/qgis/trunk/qgis \  
https://svn.qgis.org/repos/qgis/branches/qgis_newfeature  
svn commit -m "New feature branch"
```

K.5.4 Merge regularly from trunk to branch

When working in a branch you should regularly merge trunk into it so that your branch does not diverge more than necessary. In the top level dir of your branch, first type 'svn info' to determine the revision numbers of your branch which will produce output something like this:

```
timlinux@timlinux-desktop:~/dev/cpp/qgis_raster_transparency_branch$ svn info  
Caminho: .  
URL: https://svn.qgis.org/repos/qgis/branches/raster_transparency_branch  
Raiz do Repositorio: https://svn.qgis.org/repos/qgis  
UUID do repositório: c8812cc2-4d05-0410-92ff-de0c093fc19c  
Revisao: 6546  
Tipo de No: diretório
```

Agendado: normal

Autor da Ultima Mudanca: timlinux

Revisao da Ultima Mudanca: 6495

Data da Ultima Mudanca: 2007-02-02 09:29:47 -0200 (Sex, 02 Feb 2007)

Propriedades da Ultima Mudanca: 2007-01-09 11:32:55 -0200 (Ter, 09 Jan 2007)

The second revision number shows the revision number of the start revision of your branch and the first the current revision. You can do a dry run of the merge like this:

```
svn merge --dry-run -r 6495:6546 https://svn.qgis.org/repos/qgis/trunk/qgis
```

After you are happy with the changes that will be made do the merge for real like this:

```
svn merge -r 6495:6546 https://svn.qgis.org/repos/qgis/trunk/qgis
svn commit -m "Merged upstream changes from trunk to my branch"
```

K.6 Submitting Patches

There are a few guidelines that will help you to get your patches into QGIS easily, and help us deal with the patches that are sent to use easily.

K.6.1 Patch file naming

If the patch is a fix for a specific bug, please name the file with the bug number in it e.g. **bug777fix.diff**, and attach it to the original bug report in trac (<https://trac.osgeo.org/qgis/>).

If the bug is an enhancement or new feature, its usually a good idea to create a ticket in trac (<https://trac.osgeo.org/qgis/>) first and then attach you

K.6.2 Create your patch in the top level QGIS source dir

This makes it easier for us to apply the patches since we don't need to navigate to a specific place in the source tree to apply the patch. Also when I receive patches I usually evaluate them using kompare, and having the patch from the top level dir makes this much easier. Below is an example of you can include multiple changed files into your patch from the top level directory:

```
cd qgis
svn diff src/ui/somefile.ui src/app/somefile2.cpp > bug872fix.diff
```


K.6.3 Including non version controlled files in your patch

If your improvements include new files that don't yet exist in the repository, you should indicate to svn that they need to be added before generating your patch e.g.

```
cd qgis
svn add src/lib/somenewfile.cpp
svn diff > bug7887fix.diff
```

K.6.4 Getting your patch noticed

QGIS developers are busy folk. We do scan the incoming patches on bug reports but sometimes we miss things. Don't be offended or alarmed. Try to identify a developer to help you - using the ["Project Organigram"] and contact them asking them if they can look at your patch. If you don't get any response, you can escalate your query to one of the Project Steering Committee members (contact details also available on the ["Project Organigram"]).

K.6.5 Due Diligence

QGIS is licensed under the GPL. You should make every effort to ensure you only submit patches which are unencumbered by conflicting intellectual property rights. Also do not submit code that you are not happy to have made available under the GPL.

K.7 Obtaining SVN Write Access

Write access to QGIS source tree is by invitation. Typically when a person submits several (there is no fixed number here) substantial patches that demonstrate basic competence and understanding of C++ and QGIS coding conventions, one of the PSC members or other existing developers can nominate that person to the PSC for granting of write access. The nominator should give a basic promotional paragraph of why they think that person should gain write access. In some cases we will grant write access to non C++ developers e.g. for translators and documentors. In these cases, the person should still have demonstrated ability to submit patches and should ideally have submitted several substantial patches that demonstrate their understanding of modifying the code base without breaking things, etc.

K.7.1 Procedure once you have access

Checkout the sources:

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

Build the sources (see INSTALL document for proper detailed instructions)

```
cd qgis
mkdir build
ccmake ..      (set your preferred options)
make
make install  (maybe you need to do with sudo / root perms)
```

Make your edits

```
cd ..
```

Make your changes in sources. Always check that everything compiles before making any commits. Try to be aware of possible breakages your commits may cause for people building on other platforms and with older / newer versions of libraries.

Add files (if you added any new files). The svn status command can be used to quickly see if you have added new files.

```
svn status src/plugin/grass/modules
```

Files listed with ? in front are not in SVN and possibly need to be added by you:

```
svn add src/plugin/grass/modules/foo.xml
```

Commit your changes

```
svn commit src/plugin/grass/modules/foo.xml
```

Your editor (as defined in \$EDITOR environment variable) will appear and you should make a comment at the top of the file (above the area that says 'dont change this'. Put a descriptive comment and rather do several small commits if the changes across a number of files are unrelated. Conversely we prefer you to group related changes into a single commit.

Save and close in your editor. The first time you do this, you should be prompted to put in your username and password. Just use the same ones as your trac account.

L Unit Testing

As of November 2007 we require all new features going into trunk to be accompanied with a unit test. Initially we have limited this requirement to `qgis_core`, and we will extend this requirement to other parts of the code base once people are familiar with the procedures for unit testing explained in the sections that follow.

L.1 The QGIS testing framework - an overview

Unit testing is carried out using a combination of `QTestLib` (the Qt testing library) and `CTest` (a framework for compiling and running tests as part of the CMake build process). Lets take an overview of the process before I delve into the details:

- **There is some code you want to test**, e.g. a class or function. Extreme programming advocates suggest that the code should not even be written yet when you start building your tests, and then as you implement your code you can immediately validate each new functional part you add with your test. In practice you will probably need to write tests for pre-existing code in QGIS since we are starting with a testing framework well after much application logic has already been implemented.
- **You create a unit test.** This happens under `<QGIS Source Dir>/tests/src/core` in the case of the core lib. The test is basically a client that creates an instance of a class and calls some methods on that class. It will check the return from each method to make sure it matches the expected value. If any one of the calls fails, the unit will fail.
- **You include `QtTestLib` macros in your test class.** This macro is processed by the Qt meta object compiler (`moc`) and expands your test class into a runnable application.
- **You add a section to the `CMakeLists.txt` in your tests directory that will build your test.**
- **You ensure you have `ENABLE_TESTING` enabled in `ccmake / cmake` setup.** This will ensure your tests actually get compiled when you type `make`.
- **You optionally add test data to `<QGIS Source Dir>/tests/testdata`** if your test is data driven (e.g. needs to load a shapefile). These test data should be as small as possible and wherever possible you should use the existing datasets already there. Your tests should never modify this data in situ, but rather may a temporary copy somewhere if needed.
- **You compile your sources and install.** Do this using normal `make && (sudo) make install` procedure.
- **You run your tests.** This is normally done simply by doing `make test` after the `make install` step, though I will explain other approaches that offer more fine grained control over running tests.

Right with that overview in mind, I will delve into a bit of detail. I've already done much of the configuration for you in CMake and other places in the source tree so all you need to do are the easy bits - writing unit tests!

L.2 Creating a unit test

Creating a unit test is easy - typically you will do this by just creating a single .cpp file (not .h file is used) and implement all your test methods as public methods that return void. I'll use a simple test class for QgsRasterLayer throughout the section that follows to illustrate. By convention we will name our test with the same name as the class they are testing but prefixed with 'Test'. So our test implementation goes in a file called testqgsrasterlayer.cpp and the class itself will be TestQgsRasterLayer. First we add our standard copyright banner:

```

/*****
  testqgsvectorfilewriter.cpp
  -----
  Date           : Frida Nov 23 2007
  Copyright      : (C) 2007 by Tim Sutton
  Email         : tim@linfiniti.com
  *****/
*
* This program is free software; you can redistribute it and/or modify *
* it under the terms of the GNU General Public License as published by *
* the Free Software Foundation; either version 2 of the License, or    *
* (at your option) any later version.                                   *
*
*/
*****/

```

Next we use start our includes needed for the tests we plan to run. There is one special include all tests should have:

```
#include <QtTest>
```

Beyond that you just continue implementing your class as per normal, pulling in whatever headers you may need:

```

//Qt includes...
#include <QObject>
#include <QString>
#include <QObject>
#include <QApplication>
#include <QFileInfo>
#include <QDir>

//qgis includes...
#include <qgsrasterlayer.h>
#include <qgsrasterbandstats.h>
#include <qgsapplication.h>

```

Since we are combining both class declaration and implementation in a single file the class declaration comes next. We start with our doxygen documentation. Every test case should be properly documented. We use the doxygen **ingroup** directive so that all the UnitTests appear as a module in the generated Doxygen documentation. After that comes a short description of the unit test:

```
/** \ingroup UnitTests
 * This is a unit test for the QgsRasterLayer class.
 */
```

The class **must** inherit from `QObject` and include the `Q_OBJECT` macro.

```
class TestQgsRasterLayer: public QObject
{
    Q_OBJECT;
```

All our test methods are implemented as **private slots**. The `QtTest` framework will sequentially call each private slot method in the test class. There are four 'special' methods which if implemented will be called at the start of the unit test (**initTestCase**), at the end of the unit test (**cleanupTestCase**). Before each test method is called, the **init()** method will be called and after each test method is called the **cleanup()** method is called. These methods are handy in that they allow you to allocate and cleanup resources prior to running each test, and the test unit as a whole.

```
private slots:
    // will be called before the first testfunction is executed.
    void initTestCase();
    // will be called after the last testfunction was executed.
    void cleanupTestCase(){};
    // will be called before each testfunction is executed.
    void init(){};
    // will be called after every testfunction.
    void cleanup();
```

Then come your test methods, all of which should take **no parameters** and should **return void**. The methods will be called in order of declaration. I am implementing two methods here which illustrates to types of testing. In the first case I want to generally test the various parts of the class are working, I can use a **functional testing** approach. Once again, extreme programmers would advocate writing these tests **before** implementing the class. Then as you work your way through your class implementation you iteratively run your unit tests. More and more test functions should complete successfully as your class implementation work progresses, and when the whole unit test passes, your new class is done and is now complete with a repeatable way to validate it.

Typically your unit tests would only cover the **public** API of your class, and normally you do not need to write tests for accessors and mutators. If it should happen that an accessor or mutator is not working as expected you would normally implement a **regression** test to check for this (see lower down).

```
//  
// Functional Testing  
//  
  
/** Check if a raster is valid. */  
void isValid();  
  
// more functional tests here ...
```

Next we implement our **regression tests**. Regression tests should be implemented to replicate the conditions of a particular bug. For example I recently received a report by email that the cell count by rasters was off by 1, throwing off all the statistics for the raster bands. I opened a bug (ticket #832) and then created a regression test that replicated the bug using a small test dataset (a 10x10 raster). Then I ran the test and ran it, verifying that it did indeed fail (the cell count was 99 instead of 100). Then I went to fix the bug and reran the unit test and the regression test passed. I committed the regression test along with the bug fix. Now if anybody breaks this in the source code again in the future, we can immediately identify that the code has regressed. Better yet before committing any changes in the future, running our tests will ensure our changes dont have unexpected side effects - like breaking existing functionality.

There is one more benefit to regression tests - they can save you time. If you ever fixed a bug that involved making changes to the source, and then running the application and performing a series of convoluted steps to replicate the issue, it will be immediately apparent that simply implementing your regression test **before** fixing the bug will let you automate the testing for bug resolution in an efficient manner.

To implement your regression test, you should follow the naming convention of regression<TicketID> for your test functions. If no trac ticket exists for the regression, you should create one first. Using this approach allows the person running a failed regression test easily go and find out more information.

```
//  
// Regression Testing  
//  
  
/** This is our second test case...to check if a raster  
    reports its dimensions properly. It is a regression test  
    for ticket #832 which was fixed with change r7650.  
    */  
void regression832();  
  
// more regression tests go here ...
```

Finally in our test class declaration you can declare privately any data members and helper methods your unit test may need. In our case I will declare a `QgsRasterLayer *` which can be used by any of our test methods. The raster layer will be created in the `initTestCase()` function which is run before any other tests, and then destroyed using `cleanupTestCase()` which is run after all tests. By declaring helper methods (which may be called by various test functions) privately, you can ensure that they won't be automatically run by the `QTest` executable that is created when we compile our test.

```
private:
    // Here we have any data structures that may need to
    // be used in many test cases.
    QgsRasterLayer * mpLayer;
};
```

That ends our class declaration. The implementation is simply inlined in the same file lower down. First our init and cleanup functions:

```
void TestQgsRasterLayer::initTestCase()
{
    // init QGIS's paths - true means that all path will be inited from prefix
    QString qgisPath = QCoreApplication::applicationDirPath ();
    QCoreApplication::setPrefixPath(qgisPath, TRUE);
#ifdef Q_OS_LINUX
    QCoreApplication::setPkgDataPath(qgisPath + "../share/qgis");
#endif
    //create some objects that will be used in all tests...

    std::cout << "Prefix PATH: " << QCoreApplication::prefixPath().toLocal8Bit().data() \
    << std::endl;
    std::cout << "Plugin PATH: " << QCoreApplication::pluginPath().toLocal8Bit().data() \
    << std::endl;
    std::cout << "PkgData PATH: " << QCoreApplication::pkgDataPath().toLocal8Bit().data() \
    << std::endl;
    std::cout << "User DB PATH: " << QCoreApplication::qgisUserDbFilePath().toLocal8Bit() \
    .data() << std::endl;

    //create a raster layer that will be used in all tests...
    QString myFileName (TEST_DATA_DIR); //defined in CmakeLists.txt
    myFileName = myFileName + QDir::separator() + "tenbytenraster.asc";
    QFileInfo myRasterFileInfo ( myFileName );
    mpLayer = new QgsRasterLayer ( myRasterFileInfo.filePath(),
```



```
        myRasterFileInfo.completeBaseName() );
}

void TestQgsRasterLayer::cleanupTestCase()
{
    delete mpLayer;
}
```

The above init function illustrates a couple of interesting things.

1. I needed to manually set the QGIS application data path so that resources such as srs.db can be found properly. 2. Secondly, this is a data driven test so we needed to provide a way to generically locate the 'tenbytenraster.asc' file. This was achieved by using the compiler define **TEST_DATA_PATH**. The define is created in the CMakeLists.txt configuration file under <QGIS Source Root>/tests/CMakeLists.txt and is available to all QGIS unit tests. If you need test data for your test, commit it under <QGIS Source Root>/tests/testdata. You should only commit very small datasets here. If your test needs to modify the test data, it should make a copy of it first.

Qt also provides some other interesting mechanisms for data driven testing, so if you are interested to know more on the topic, consult the Qt documentation.

Next let's look at our functional test. The isValid() test simply checks the raster layer was correctly loaded in the initTestCase. QVERIFY is a Qt macro that you can use to evaluate a test condition. There are a few other use macros Qt provide for use in your tests including:

```
QCOMPARE ( actual, expected )
QEXPECT_FAIL ( dataIndex, comment, mode )
QFAIL ( message )
QFETCH ( type, name )
QSKIP ( description, mode )
QTEST ( actual, testElement )
QTEST_APPLESS_MAIN ( TestClass )
QTEST_MAIN ( TestClass )
QTEST_NOOP_MAIN ( )
QVERIFY2 ( condition, message )
QVERIFY ( condition )
QWARN ( message )
```

Some of these macros are useful only when using the Qt framework for data driven testing (see the Qt docs for more detail).

```
void TestQgsRasterLayer::isValid()
```

```
{  
    QVERIFY ( mpLayer->isValid() );  
}
```

Normally your functional tests would cover all the range of functionality of your classes public API where feasible. With our functional tests out the way, we can look at our regression test example.

Since the issue in bug #832 is a misreported cell count, writing our test is simply a matter of using QVERIFY to check that the cell count meets the expected value:

```
void TestQgsRasterLayer::regression832()  
{  
    QVERIFY ( mpLayer->getRasterXDim() == 10 );  
    QVERIFY ( mpLayer->getRasterYDim() == 10 );  
    // regression check for ticket #832  
    // note getRasterBandStats call is base 1  
    QVERIFY ( mpLayer->getRasterBandStats(1).elementCountInt == 100 );  
}
```

With all the unit test functions implemented, there one final thing we need to add to our test class:

```
QTEST_MAIN(TestQgsRasterLayer)  
#include "moc_testqgsrasterlayer.cxx"
```

The purpose of these two lines is to signal to Qt's moc that this is a QtTest (it will generate a main method that in turn calls each test function). The last line is the include for the MOC generated sources. You should replace 'testqgsrasterlayer' with the name of your class in lower case.

L.3 Adding your unit test to CMakeLists.txt

Adding your unit test to the build system is simply a matter of editing the CMakeLists.txt in the test directory, cloning one of the existing test blocks, and then search and replacing your test class name into it. For example:

```
#  
# QgsRasterLayer test  
#  
SET(qgis_rasterlayertest_SRCS testqgsrasterlayer.cpp)  
SET(qgis_rasterlayertest_MOC_CPPS testqgsrasterlayer.cpp)
```

```

QT4_WRAP_CPP(qgis_rasterlayertest_MOC_SRCS ${qgis_rasterlayertest_MOC_CPPS})
ADD_CUSTOM_TARGET(qgis_rasterlayertestmoc ALL DEPENDS ${qgis_rasterlayertest_MOC_SRCS})
ADD_EXECUTABLE(qgis_rasterlayertest ${qgis_rasterlayertest_SRCS})
ADD_DEPENDENCIES(qgis_rasterlayertest qgis_rasterlayertestmoc)
TARGET_LINK_LIBRARIES(qgis_rasterlayertest ${QT_LIBRARIES} qgis_core)
INSTALL(TARGETS qgis_rasterlayertest RUNTIME DESTINATION ${QGIS_BIN_DIR})
ADD_TEST(qgis_rasterlayertest ${QGIS_BIN_DIR}/qgis_rasterlayertest)

```

I'll run through these lines briefly to explain what they do, but if you are not interested, just clone the block, search and replace e.g.

```
: '<, '>s/rasterlayer/mynewtest/g
```

Lets look a little more in detail at the individual lines. First we define the list of sources for our test. Since we have only one source file (following the methodology I described above where class declaration and definition are in the same file) its a simple statement:

```
SET(qgis_rasterlayertest_SRCS testqgsrasterlayer.cpp)
```

Since our test class needs to be run through the Qt meta object compiler (moc) we need to provide a couple of lines to make that happen too:

```

SET(qgis_rasterlayertest_MOC_CPPS testqgsrasterlayer.cpp)
QT4_WRAP_CPP(qgis_rasterlayertest_MOC_SRCS ${qgis_rasterlayertest_MOC_CPPS})
ADD_CUSTOM_TARGET(qgis_rasterlayertestmoc ALL DEPENDS ${qgis_rasterlayertest_MOC_SRCS})

```

Next we tell cmake that it must make an executable from the test class. Remember in the previous section on the last line of the class implementation I included the moc outputs directly into our test class, so that will give it (among other things) a main method so the class can be compiled as an executable:

```

ADD_EXECUTABLE(qgis_rasterlayertest ${qgis_rasterlayertest_SRCS})
ADD_DEPENDENCIES(qgis_rasterlayertest qgis_rasterlayertestmoc)

```

Next we need to specify any library dependencies. At the moment classes have been implemented with a catch-all QT_LIBRARIES dependency, but I will be working to replace that with the specific Qt libraries that each class needs only. Of course you also need to link to the relevant qgis libraries as required by your unit test.

```
TARGET_LINK_LIBRARIES(qgis_rasterlayertest ${QT_LIBRARIES} qgis_core)
```

Next I tell cmake to the same place as the qgis binaries itself. This is something I plan to remove in the future so that the tests can run directly from inside the source tree.

```
INSTALL(TARGETS qgis_rasterlayertest RUNTIME DESTINATION ${QGIS_BIN_DIR})
```

Finally here is where the best magic happens - we register the class with ctest. If you recall in the overview I gave in the beginning of this section we are using both QTest and CTest together. To recap, **QTest** adds a main method to your test unit and handles calling your test methods within the class. It also provides some macros like QVERIFY that you can use as to test for failure of the tests using conditions. The output from a QTest unit test is an executable which you can run from the command line. However when you have a suite of tests and you want to run each executable in turn, and better yet integrate running tests into the build process, the **CTest** is what we use. The next line registers the unit test with CMake / CTest.

```
ADD_TEST(qgis_rasterlayertest ${QGIS_BIN_DIR}/qgis_rasterlayertest)
```

The last thing I should add is that if your test requires optional parts of the build process (e.g. Postgresql support, GSL libs, GRASS etc.), you should take care to enclose you test block inside a IF () block in the CMakeLists.txt file.

L.4 Building your unit test

To build the unit test you need only to make sure that ENABLE_TESTS=true in the cmake configuration. There are two ways to do this:

1. Run `ccmake ..` (`cmakesetup ..` under windows) and interactively set the ENABLE_TESTS flag to ON.
1. Add a command line flag to cmake e.g. `cmake -DENABLE_TESTS=true ..`

Other than that, just build QGIS as per normal and the tests should build too.

L.5 Run your tests

The simplest way to run the tests is as part of your normal build process:

```
make && make install && make test
```

The make test command will invoke CTest which will run each test that was registered using the ADD_TEST CMake directive described above. Typical output from make test will look like this:

```
Running tests...
Start processing tests
Test project /Users/tim/dev/cpp/qgis/build
1/ 3 Testing qgis_applicationtest      ***Exception: Other
2/ 3 Testing qgis_filewritertest      *** Passed
3/ 3 Testing qgis_rasterlayertest     *** Passed
```

0% tests passed, 3 tests failed out of 3

```
The following tests FAILED:
1 - qgis_applicationtest (OTHER_FAULT)
Errors while running CTest
make: *** [test] Error 8
```

If a test fails, you can use the `ctest` command to examine more closely why it failed. Use the `-R` option to specify a regex for which tests you want to run and `-V` to get verbose output:

```
[build] ctest -R appl -V
Start processing tests
Test project /Users/tim/dev/cpp/qgis/build
Constructing a list of tests
Done constructing a list of tests
Changing directory into /Users/tim/dev/cpp/qgis/build/tests/src/core
1/ 3 Testing qgis_applicationtest
Test command: /Users/tim/dev/cpp/qgis/build/tests/src/core/qgis_applicationtest
***** Start testing of TestQgsApplication *****
  Config: Using QTest library 4.3.0, Qt 4.3.0
PASS   : TestQgsApplication::initTestCase()
  Prefix PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/./
  Plugin  PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/./lib/qgis
  PkgData PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/./share/qgis
  User DB PATH: /Users/tim/.qgis/qgis.db
PASS   : TestQgsApplication::getPaths()
  Prefix PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/./
  Plugin  PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/./lib/qgis
  PkgData PATH: /Users/tim/dev/cpp/qgis/build/tests/src/core/./share/qgis
  User DB PATH: /Users/tim/.qgis/qgis.db
QDEBUG : TestQgsApplication::checkTheme() Checking if a theme icon exists:
QDEBUG : TestQgsApplication::checkTheme()
/Users/tim/dev/cpp/qgis/build/tests/src/core/.. \
//share/qgis/themes/default/mIconProjectionDisabled.png
```

M HIG (HUMAN INTERFACE GUIDELINES)

```
FAIL! : TestQgsApplication::checkTheme() '!myPixmap.isNull()' returned FALSE. ()
Loc: [/Users/tim/dev/cpp/qgis/tests/src/core/testqgsapplication.cpp(59)]
PASS : TestQgsApplication::cleanupTestCase()
Totals: 3 passed, 1 failed, 0 skipped
***** Finished testing of TestQgsApplication *****
-- Process completed
***Failed

0% tests passed, 1 tests failed out of 1
```

```
The following tests FAILED:
1 - qgis_applicationtest (Failed)
Errors while running CTest
```

Well that concludes this section on writing unit tests in QGIS. We hope you will get into the habit of writing test to test new functionality and to check for regressions. Some aspects of the test system (in particular the CMakeLists.txt parts) are still being worked on so that the testing framework works in a truly platform way. I will update this document as things progress.

M HIG (Human Interface Guidelines)

In order for all graphical user interface elements to appear consistent and to all the user to instinctively use dialogs, it is important that the following guidelines are followed in layout and design of GUIs.

1. Group related elements using group boxes: Try to identify elements that can be grouped together and then use group boxes with a label to identify the topic of that group. Avoid using group boxes with only a single widget / item inside.
2. Capitalise first letter only in labels: Labels (and group box labels) should be written as a phrase with leading capital letter, and all remaining words written with lower case first letters
3. Do not end labels for widgets or group boxes with a colon: Adding a colon causes visual noise and does not impart additional meaning, so don't use them. An exception to this rule is when you have two labels next to each other e.g.: Label1 **Plugin** Label2 [/path/to/plugins]
4. Keep harmful actions away from harmless ones: If you have actions for 'delete', 'remove' etc, try to impose adequate space between the harmful action and innocuous actions so that the users is less likely to inadvertently click on the harmful action.
5. Always use a QDialogBox for 'OK', 'Cancel' etc buttons: Using a button box will ensure that the order of 'OK' and 'Cancel' etc, buttons is consistent with the operating system / locale / desktop environment that the user is using.

N GNU General Public License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 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.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, 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 or use pieces of it in new free programs; and that you know you can do these things.

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

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow. **TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's 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 give any other recipients of the Program a copy of this License along with the Program.

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 Program or any portion of it, thus forming a work based on the Program, 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) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, 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 Program, 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 Program.

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

the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) 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; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, 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 executable. However, as a special exception, the source code 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.

If distribution of executable or 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 counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program 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.

5. 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 Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program 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 to this License.

7. 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 Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program 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 Program.

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.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program 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.

9. The Free Software Foundation may publish revised and/or new versions of the 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 Program 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 Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, 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

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "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 PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. 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 PRO-

GRAM 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 PROGRAM (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 PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

N.1 Quantum GIS Qt exception for GPL

In addition, as a special exception, the QGIS Development Team gives permission to link the code of this program with the Qt library, including but not limited to the following versions (both free and commercial): Qt/Non-commercial Windows, Qt/Windows, Qt/X11, Qt/Mac, and Qt/Embedded (or with modified versions of Qt that use the same license as Qt), and distribute linked combinations including the two. You must obey the GNU General Public License in all respects for all of the code used other than Qt. If you modify this file, you may extend this exception to your version of the file, but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version.

Literature

- [1] T. Mitchell. Web mapping illustrated, published by o'reilly, 2005.
- [2] M. Neteler and H. Mitasova. Open source gis: A grass gis approach. 3. edition, springer, new york, 2008.

Web-References

- [3] GRASS GIS. <http://grass.osgeo.org>, 2008.
- [4] PostGIS. <http://postgis.refractory.net/>, 2006.
- [5] Web Map Service (1.1.1) Implementation Specification. <http://portal.opengeospatial.org>, 2002.
- [6] Web Map Service (1.3.0) Implementation Specification. <http://portal.opengeospatial.org>, 2004.