Shuffling Quantum GIS into the Open Source GIS Stack

Free and Open Source Software for Geospatial (FOSS4G) Conference
Victoria, BC

Gary E. Sherman

September 24, 2007

Contents

Tips

Exercises

Chapter 1

Introduction

The goal of this workshop is to not only introduce you to Quantum GIS (QGIS), but to show you how it can integrate with other members of the open source GIS stack.

1.1 Topics

Here are the topics we will cover:

1. Install QGIS
2. View common file-based data
3. Symbolize your data
4. Work with attributes and actions
5. Digitize features
6. Export map images
7. Create data from text files
8. Use the standard plugins that come with QGIS
9. Export MapServer map files
10. Consume data from WMS and WFS servers
11. View and edit data in PostGIS
12. View, edit, and analyze GRASS vector and raster maps

We will also take a look at using Python to extend QGIS, including how you can write your own applications using Python and PyQt.

1.2 Tools

For the workshop, we will be using a LiveCD. If you brought your own laptop, you can run the LiveCD on it (assuming it's an i386 machine) or you can use pre-installed software you may have already. We will be using the following applications:

1. QGIS 0.8.1
2. GRASS 6.2 or higher
3. PostgreSQL with PostGIS
4. QGIS 0.9.0

1.3 Data

For the workshop we will be using a dataset found on the LiveCD. If you wish to use your own laptop and software for the workshop, you can install the data from one of several CDs we have available.

1.4 Conventions

In the text you will find several conventions to indicate certain items and actions:

- Close - A button on a dialog
- **QInputDialog** - A class name
- *LENGTH* - A field in a layer or table
- `cities.shp` - A file or directory name
- Enter - A key on the keyboard
- *getName* - A class method
- File - A Menu Item
- *localhost* - A server name
- **geometry_columns** - A table in a database
- Open Table - A Tool on the Toolbar
- `Text you type` - Text you type

Chapter 2

Getting Started

In this chapter we'll look at the basics of QGIS, including viewing and symbolizing data, as well as working with attributes and actions. We'll be using the LiveCD; if you want help installing QGIS, see Appendix A for information specific to your platform.

2.1 Using the LiveCD

To use the LiveCD, you must reboot:

The LiveCD is based on Gentoo and uses the Gnome desktop

1. Insert the LiveCD into the CD-ROM drive
2. If the computer is running, reboot it using the Start menu
3. If the computer is off, turn it on
4. When the boot prompt appears, press Enter
5. When prompted for the keymap, press Enter
6. Wait while the system comes up
7. At the login screen, enter `qgisuser` as the user name
8. Enter `qgisuser` as the password

If everything goes well, you should be looking at a Linux Gnome desktop.

2.2 Starting QGIS

To start QGIS, choose Qgis from the Applications/Gis menu. You should now be looking at the QGIS map canvas and an empty legend.

2.3 Viewing Data

Let's get started viewing some data by loading some vector and raster datasets.

2.3.1 Viewing Vector Data

Exercise 1 Loading the World Borders Layer

We'll start with vector data and load the world borders layer:

1. Click on the (Add a Vector Layer) tool in the toolbar or choose Add a Vector Layer from the Layer menu.
2. Navigate to the data directory (`/home/qgisuser/data/qgis`)
3. Make sure the "Files of type" drop-down box has the "ESRI Shapefiles" filter selected
4. Choose `world_borders.shp`
5. Click Open

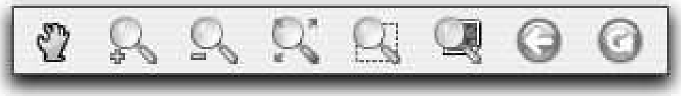
Tip 1 Loading More than One Layer

You can load more than one layer at a time by using the Shift or Ctrl keys in the open dialog to select multiple layers.

You should now have the `world_borders` shapefile loaded and displayed in an ugly color. We'll fix that in just a moment.

Using the same steps, load the `cities.shp` layer from the data directory.

Once you have the layers loaded, take a look at the map navigation toolbar



From left to right, the tools are:

1. Pan - pan the map by dragging the mouse
2. Zoom In - zoom in by dragging the mouse or clicking
3. Zoom Out - zoom out by dragging the mouse or clicking
4. Zoom Full - zoom to the full extent of all layers
5. Zoom to Selection - zoom to the extent of the selected features
6. Zoom to Layer - zoom to the extent of the active layer
7. Zoom to Last Extent - zoom to the previous view
8. Refresh - redraw the map

Feel free to experiment with the map navigation tools.

Tip 2 MISSING TOOLBARS

If you find one or more of the toolbars are missing, right-click in the toolbar area to bring up the popup menu. Click on the toolbars that you want to make visible.

2.3.2 Viewing Raster Data

Exercise 2 Loading the World Mosaic Raster

Let's add the `world_mosaic.tif` raster to our map.

1. Click on the (Add a Raster Layer) tool in the toolbar or choose Add a Raster Layer from the Layer menu.
2. Navigate to the data directory (`/home/qgisuser/data/qgis`)
3. Make sure the "Files of type" drop-down box has the "GeoTIFF" filter selected
4. Choose `world_mosaic.tif`
5. Click [Open]

The raster is now loaded over your vector data. You can rearrange the layers by clicking on them in the legend and dragging them to a new position in the stack. Drag the `cities` layer to the top of the stack so the cities are displayed over the raster.

Don't worry about the ugliness—we'll fix that next.

2.4 Symbolizing Your Data

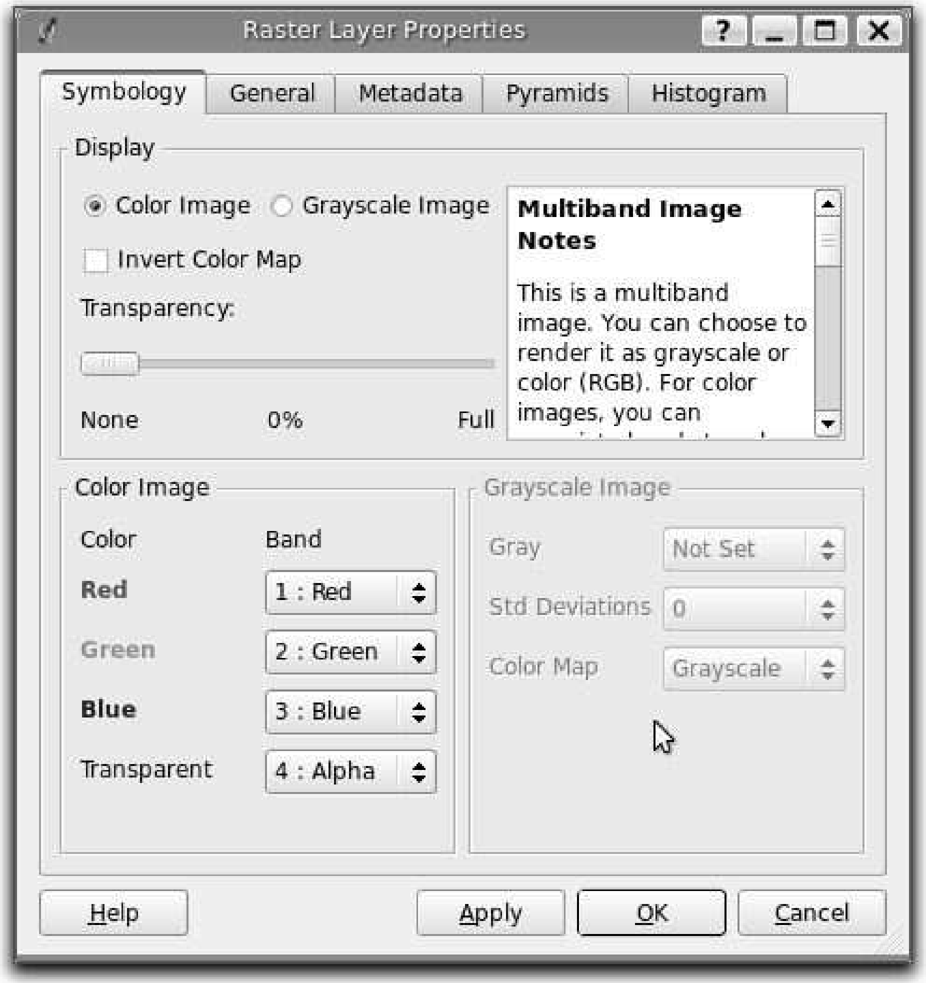
QGIS gives you a good selection of rendering options for your raster and vector data. These options are accessed using the properties dialog for a layer. We'll start with the raster first.

2.4.1 Raster Rendering

Exercise 3 Changing Raster Transparency

The raster we loaded has an alpha (transparency) channel. When loaded into QGIS, the oceans are set to transparent, giving things a ragged look. To get the oceans back:

1. Double-click on the `world_mosaic` layer in the legend to open the raster properties dialog.
2. In the "Transparent" drop-down box, change the value from "Alpha" to "Not Set"
3. Click OK



Display

● Color Image ○ Grayscale Image

□ Invert Color Map

Transparency:

None 0% Full

Multiband Image Notes

This is a multiband image. You can choose to render it as grayscale or color (RGB). For color images, you can

Color Image

Color	Band
Red	1 : Red
Green	2 : Green
Blue	3 : Blue
Transparent	4 : Alpha

Grayscale Image

Gray	Not Set
Std Deviations	0
Color Map	Grayscale

The oceans have returned. Let's reopen the raster properties dialog and take a quick look at the options on each of the tabs.

Symbology Tab

In addition to changing the band settings for the Red, Green, Blue, and Alpha bands, you can also do a number of things related to symbology:

1. Change between color and grayscale
2. Invert the color map
3. Set the transparency for the raster
4. Adjust the settings for a grayscale image

Be sure to read the image notes for information about your raster.

Tip 3 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". In version 0.8.1, this causes QGIS to crash. To display the Red band, set the image type to grayscale, then select Red as the band to use for Gray.

General Tab

Hint - Use the scale display on the toolbar to help in setting scale dependency values

The general tab allows you to:

1. Change the name of the layer as it appears in the legend
2. Set scale dependency so the layer is only visible between and minimum and/or maximum scale
3. View or change the spatial reference (projection)
4. View the thumbnail, legend, and palette graphics

Typically you'll use this most frequently to change the display name and set scale dependency.

Metadata Tab

The Metadata tab provides more information than you would want to know about your raster, including the size, number of bands, data type, projection, pixel size, and band properties.

Pyramids Tab

Using pyramids helps speed up the display of a raster at smaller scales (zoomed out). The Pyramids tab allows you to build pyramids for a raster. Make sure to read the warnings as this process will alter your original image. The available pyramids for your raster are displayed in the right-hand panel. If a pyramid is available for a given resolution, the little pyramid icon will be present (without a red X over it).

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 right of the dialog.

Once you view the histogram, you'll notice that the band statistics have been populated on the Metadata tab.

2.4.2 Vector Rendering

QGIS provides four renderers to symbolize your data:

Single Symbol
This uses a single symbol for point layers, a simple outline for line layers, and a fill color for polygons. All features in the layer are rendered the same way.

Graduated Symbol
This renderer allows you to symbolize features based on a number of classes. Each class is composed of a range of values. The field used to render the features must be numeric.

Continuous Color
This renderer renders the features based on the value of a field, starting with a color for the minimum value and ending with a color for the maximum value. The field used to classify the data must be numeric.

Unique Value
This renderer displays all features of the same type in the same color. You can use any field in the attribute table as the classification field. For each unique value, you must set the color and fill style, or accept the random defaults assigned by QGIS.

The choice of renderer depends on what you are trying to visualize. Here are some examples:

1. You want to display world population to portray the most populated countries - Graduated Symbol
2. You want to create a map showing rock types (or soil, vegetation, or any other thing) - Unique Value
3. You want to render earthquakes by magnitude and aren't too concerned with the class breaks - Continuous Color

To use a renderer, select if from the "Legend Type" drop-down list. When you select a renderer, the dialog will change to provide the needed controls for setting things up. These are different for each renderer.

Exercise 4 Rendering Cities by Capital vs Non-Capital

Take a shot at rendering the `cities` layer to display capital cities in a different color or style than non-capital cities. While you're at it, fix the ugly random color for your `world_borders` layer.

Let's take a look at the vector properties dialog along with some of the options and tabs.

Symbology Tab

We've discussed the Symbology tab in the previous section. In addition to choosing a renderer, the tab allows you to set the transparency for the layer. When using a renderer on this dialog, you can change the colors for a given class or property by clicking on the associated color box to popup the color selector.

The best way to get acquainted with the renderers and their capabilities is to experiment with them.

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, and view or change the projection.

The [Query Builder] button allows you to create a subset of the features in the layer—but this only works for PostGIS layers.

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 projection and attribute fields and their data type are displayed on this tab. This is a quick way to get information about the layer.

Labels Tab

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

Exercise 5 Labeling Cities

Let's illustrate by labelling the cities on our map:

1. Zoom in a bit to your favorite continent
2. Make the `cities` layer active
3. Open the properties dialog
4. Click on the "Labels" tab
5. Check the "Display labels" checkbox to enable labeling
6. Choose the field to label with. We'll use *NAME*
7. Enter a default for cities that have no name. The default label will be used each time QGIS encounters a city with no value in the *NAME* field.
8. 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 city.

Click on the "Font Style" tab and use the [Font] and [Colour] buttons to set the font and color.

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

1. Click on the "Font Alignment" tab
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. Click Apply to see your changes without closing the dialog

Things are looking better, but the label is still too close to the marker. To fix this we can use the options on the "Position" tab. 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 city 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 Colour and choosing your favorite from the color selector
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 2 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 tabs on the Label tab allow you control the appearance of the labels using attributes stored in the layer. The "Data" tabs allow you to set all the parameters for the labels using fields in the layer.

Actions Tab

We'll take a close look at Attribute Actions in the next section.

2.5 Attributes and Actions

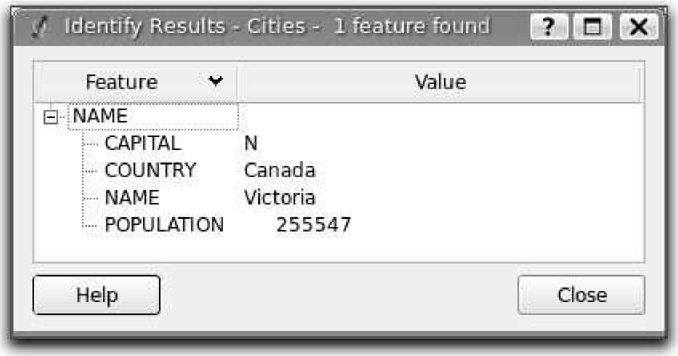
QGIS provides several methods to view and work with the attributes of your vector data. The simplest is the Identify function.

2.5.1 Identifying Features

To identify a feature on a layer:

1. Make the layer active by clicking on its name in the legend
2. Click on the Identify Features tool
3. Click on a feature

⋆⋆**Try It** Try this with the `cities` layer. If you click on or close enough to a city, you'll get a results dialog similar to this:



Feature	Value
NAME	
CAPITAL	N
COUNTRY	Canada
NAME	Victoria
POPULATION	255547

The fields in the `cities` layer are displayed, along with the corresponding value for each. If you click among a cluster of features, you may see more than one result in the dialog.

If you keep getting "No features found" when trying to identify a feature you can do one of two things (or both):

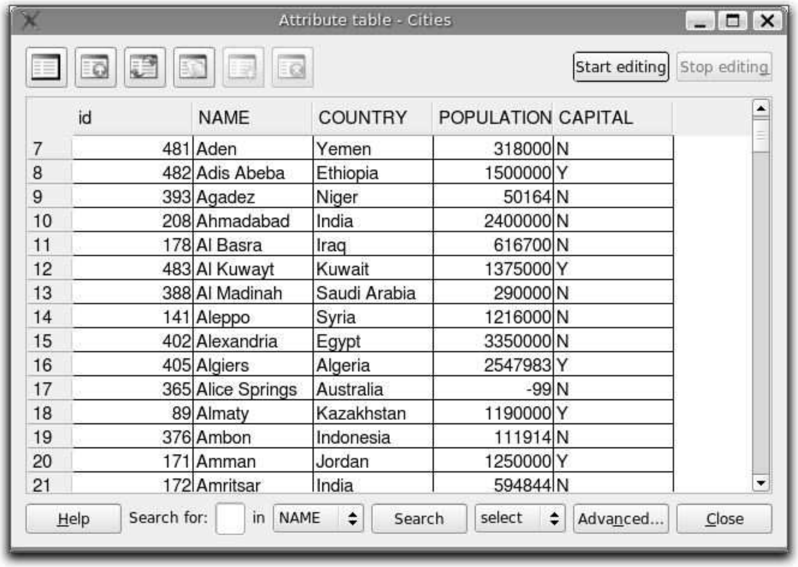
- Zoom in and try again
- Change the search radius used to find a feature

Exercise 6 Changing the Search Radius

To change the search radius, choose Options... from the Settings menu, then click on the "Map tools" tab. The search radius is specified as a percentage of the map width. If you aren't able to click on a feature, try increasing it to something like 0.7%. If you find that too many features are being returned, decrease the search radius.

2.5.2 Using the Attribute Table

To open the attribute table for a layer, click on the (Open Table) tool. The attribute table contains a row for each feature in your layer:



	id	NAME	COUNTRY	POPULATION	CAPITAL
7	481	Aden	Yemen	318000	N
8	482	Adis Abeba	Ethiopia	1500000	Y
9	393	Agadez	Niger	50164	N
10	208	Ahmadabad	India	2400000	N
11	178	Al Basra	Iraq	616700	N
12	483	Al Kuwayt	Kuwait	1375000	Y
13	388	Al Madinah	Saudi Arabia	290000	N
14	141	Aleppo	Syria	1216000	N
15	402	Alexandria	Egypt	3350000	N
16	405	Algiers	Algeria	2547983	Y
17	365	Alice Springs	Australia	-99	N
18	89	Almaty	Kazakhstan	1190000	Y
19	376	Ambon	Indonesia	111914	N
20	171	Amman	Jordan	1250000	Y
21	172	Amritsar	India	594844	N

Besides just viewing the attributes for your layer, you can use the attribute table dialog to:

- Select features by highlighting them. They will be highlighted on the map as well
- Once you have selected some rows, you can move them to the top of the table, invert the selection, or copy it to the clipboard using the tools in the toolbar at the top of the dialog.
- Sort rows by any of the field (column) names
- Edit attributes
- Search the table by any field
- Use the advanced query builder to perform more complex searches

The dialog also allows you to create or delete a new column, but this functionality only works with PostGIS layers.

Editing Attributes

To edit one or more field values:

1. Click on the Start editing button
2. Locate the row and field you want to edit
3. Double-click on the field contents to enable editing
4. Edit the contents
5. Press Enter, Tab, or click on another row to make the change stick
6. When done click on Stop editing
7. A dialog confirming your changes will popup. If you want to save your edits, click Yes, otherwise click No to discard your edits.

You might have noticed that Alice Springs has a population of -99. Surely there must be more people living there. Try changing it to something more respectable. ⋆⋆**Try It**

Tip 4 EDITING ATTRIBUTES

Not all layer types can be edited. It depends on the capabilities of the underlying data provider. To see if your layer type supports editing, open the vector properties dialog and look in the "General" section of the "Metadata" tab.

Searching

To search the attribute table:

1. Enter the search string in the "Search for" box
2. Choose the field you want to search from the drop-down list
3. Change the "select" drop-down to the option you want. You can select the matches, select them and bring to the top of the table, or show only matching records.
4. Click [Search]

Query Builder

To create more complex queries, use the Query Builder by clicking on the [Advanced] button. If you have used other GIS software, the layout of the Query Builder will look familiar. You can either build the query by pointing and clicking (and a little typing) or by entering the SQL *where* clause directly.

⋆⋆**Try It** As an example, try selecting all the cities that have a population less than 100,000 but are capital cities. Once you make the selection, return to the attribute table and promote the results to the top.

2.5.3 Attribute Actions

Attribute actions are a handy way to run an external program and pass it one or more fields associated with a feature. For example, you could create actions to display an image, do a web search, or create a report from a database based on the value.

An action looks something like this: `http://en.wikipedia.org/wiki/%CNTRY_NAME`

If you create this action for a world boundaries layer that has a *CNTRY_NAME* field, you can use it to open the Wikipedia page for a country.

Let's create an action that does a Google search on the `cities` 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 `cities` layer is loaded
2. Open the 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

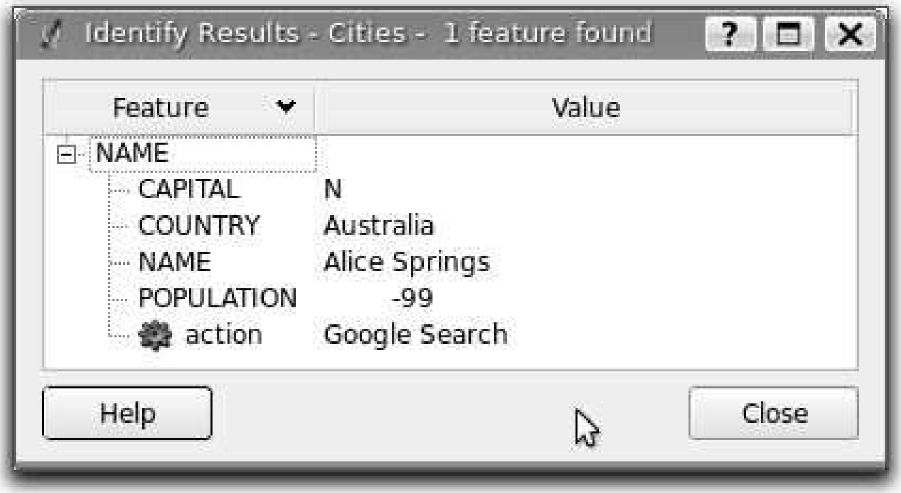
full path.

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 `cities` layer. It's located just to the right of the Insert Field button.
9. From the drop-down box, select *NAME* and click Insert Field
10. Your action text now looks like this:
    `firefox http://google.com/search?q=%NAME`
11. Add a "+" to the end of the action text
12. From the drop-down list of fields, select *COUNTRY* and click Insert Field
13. Add the action to the list for this layer by clicking Insert action

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=%NAME+%COUNTRY
```

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



Feature	Value
NAME	
CAPITAL	N
COUNTRY	Australia
NAME	Alice Springs
POPULATION	-99
action	Google Search

When we click on the action, it brings up Firefox and navigates to the URL `http://www.google.com/search?q=Alice%20Springs+Australia`. We could have just searched on *NAME*, but then certain cities we choose might give us undesirable results without the country name. I'll leave those up to your imagination.

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.

Exercise 7 Creating an Attribute Action

Create an action to do the Wikipedia lookup using the `world_borders` layer and the *CNTRY_NAME* field. The URL needed for the lookup is `http://en.wikipedia.org/wiki/`. Add the country name field to the end of the URL when creating the action.

Chapter 3

Digitizing and Editing Data

QGIS supports digitizing and editing of spatial features in a wide range of formats, including shapefiles, PostGIS layers, and GRASS layers.

3.1 Digitizing

Our goal is to do some heads-up digitizing from the `world_mosaic` raster. We'll assume we are creating a brand new layer rather than editing an existing one. As of version 0.8, QGIS only supports creation of shapefiles for editing, although you could create a new PostgreSQL layer using SQL and edit it.

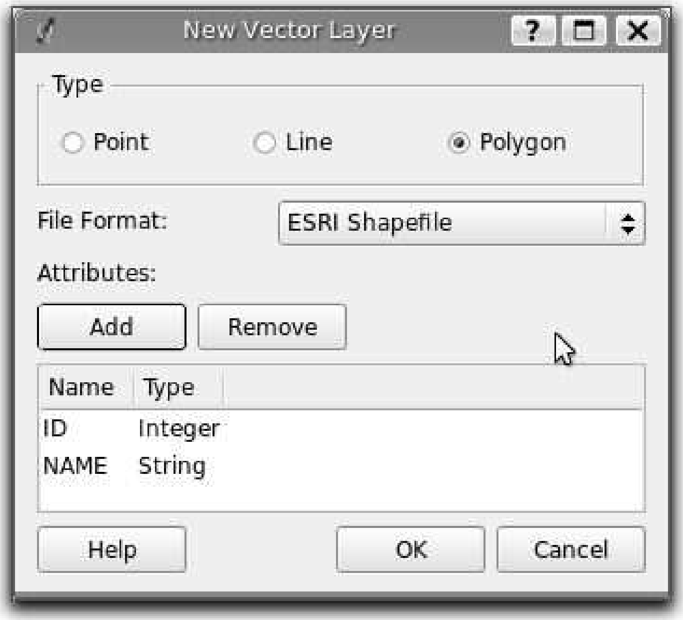
Make sure the world mosaic is loaded in QGIS

Exercise 8 Creating a New Shapefile

We'll create a shapefile with an *ID* and *NAME*

1. Choose New Vector Layer... from the Layer menu
2. Click on the "polygon" radio button
3. Click Add and add the *ID* field with type "Integer"
4. Click Add and add the *NAME* field, with type "String"

When all set, your dialog should look like this:



Type

○ Point ○ Line ● Polygon

File Format: ESRI Shapefile

Attributes:

Name	Type
ID	Integer
NAME	String

Click OK to create the layer. This opens the dialog to save the file, allowing you to navigate to the directory where you want the shapefile to live and give it an appropriate name.

Our empty shapefile is created and displayed in QGIS.

To start digitizing, we need to allow editing on the new layer by right-clicking on it in the legend and choosing Allow editing from the popup menu. A small pencil with a little scribble trailing from it appears over the layer's icon in the legend to indicate that we are now in edit mode.

Before we start editing, we need to make sure the digitizing toolbar is visible. If not, right-click in the toolbar area of QGIS and choose Digitizing from the popup menu. Some of the buttons on the toolbar are disabled (grayed out), in particular the point and line buttons. This is because QGIS knows we are editing a polygon layer and won't allow you to create the wrong feature type.

Exercise 9 Digitizing a Lake

Let's start digitizing:

1. Zoom in to an area on the `world_mosaic` where there are one or more lakes. You'll find some nice candidates in central North America, Africa, and eastern Russia. To get a gross approximation of the lake, we can just zoom in until we only see the lake. If we want a detailed representation of the lake, we need to zoom in much closer and pan around the map as we digitize.
2. If you aren't already in edit mode, click the (Start editing) tool
3. Click on the (Capture Polygon) tool in the digitizing toolbar
4. To digitize, click with the left mouse button and begin moving along the shoreline, clicking at each location where there is a change in direction
5. If you get to the edge of the map canvas and need to pan to see more of the lake, don't use the pan tool, but instead hold down the [Space] bar and move the mouse to pan.
6. If you find out that you need to change the zoom level to effectively digitize, you can zoom in and out using the mouse wheel
7. To complete the lake, right-click at the final point

This brings up the attribute dialog where we enter the attribute values for the feature. Enter an *ID* number and a *NAME* for the lake.

To actually save the changes, we need to stop editing by clicking on the (Stop editing) tool in the toolbar. Or you can right-click on the layer name in the legend and choose Allow editing to toggle editing off. A confirmation dialog appears and you have to choose [Yes] to save the edits.

The lake now appears like a regular polygon feature, and we can both identify it and view the attribute table. You can continue digitizing by starting editing again using the (Start editing) tool.

3.2 Editing Your Features

You probably noticed that as we were digitizing, we couldn't correct any mistakes we made along the way. Fortunately we can fix things up once we have completed the feature. QGIS indicates the vertices (or points) with a large "X" when a layer is in edit mode. The tools on the digitizing toolbar allow us to add, delete, and move the vertices to adjust a feature.

If you find areas where you were a bit careless or maybe should have been more detailed, don't worry—you can easily fix them.

3.2.1 Setting the Snap Tolerance

Before we can edit vertices, we need to set the snapping tolerance. This is the distance QGIS uses to "search" for the polygon and vertex you are trying to edit when you click on the map. If you aren't within the snap tolerance, QGIS won't find and select the vertex for editing. Tolerance is set in map units so you may find you need to experiment to get it 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 it too small and it won't find anything and it will pop up an annoying warning to that effect.

⋆⋆**Try It** To set the snap tolerance, choose Project Properties from the Settings menu and click on the "General" tab. Remember the tolerance is in map units. For our little digitizing project, the units are in decimal degrees. Your results may vary, but something on the order of 0.05 to 0.1 should be fine.

3.2.2 Editing Vertices

⋆⋆**Try It** Once the tolerance is set, we can move vertices to correct mistakes and better match the shoreline. Just click on the (Move Vertex) tool, place the cursor over the vertex to be moved, and drag it to the new location. When you release the mouse, the vertex is moved and the shape of the polygon changes.

⋆⋆**Try It** If you find you missed some detail (for example a straight line that missed a little inlet in the lake) you can add one or more vertices to solve the problem. First use the (Add Vertex) tool to add the vertices by clicking on the line segment that needs to be modified. Don't worry if you don't get them exact. Once you've added the vertices you need, use the move tool to adjust the positions to reflect the shoreline of the inlet. If you didn't add enough vertices to accurately portray the feature—no problem—just add some more and move them around until the job is done.

⋆⋆**Try It** You may find that you have too many vertices or just plain put one where it doesn't belong. In that case, use the (Delete Vertex) tool to get rid of the ones you don't need. When you delete a vertex the polygon reshapes itself automatically.

3.2.3 Copy, Cut, and Paste

What if we want to delete an entire polygon? We can do that by first selecting the polygon (in our example a lake) using the regular (Select Features) tool. You can select multiple features for deletion. Once you have the selection 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.

The (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.

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.

Exercise 10 Copying Features Between Layers

Try copying some of your 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.

Chapter 4

Working with Plugins

QGIS comes with 12 plugins that enhance and extend the application. The standard plugins available with QGIS are:

Add Delimited Text Layer
Loads and displays data from a delimited text file. The text file must contain X and Y coordinates for each feature. Only point data is supported by the plugin.

Copyright Label
Displays copyright information on the map canvas. You can customize the text, style and placement of the label.

GPS Tools
Tools to load and import GPS data, as well as download to your GPS unit.

GRASS
A full suite of GRASS tools for loading vector and raster maps, digitizing features, and using modules to import, export, and process data.

Georeferencer
Georeference rasters by interactively setting control points to create a world file.

Graticule Creator
Create a graticule (grid) shapefile by specifying the extent and interval between latitude and longitude lines. You can create a point, line, or polygon graticule with this plugin.

Launcher
Launches a program or script from QGIS and captures the output. Commands are stored in a drop-down list for future use.

North Arrow
Displays a customizable north arrow on the map canvas. You can adjust the placement of the arrow, as well as the angle. Or you can just let QGIS determine the angle direction automatically.

PostgreSQL Geoprocessing
Tools for processing PostGIS layers. At present this plugin is rather limited and contains only a Buffer tool.

SPIT
Shapefile to PostGIS Import Tool allows you to import shapefiles into PostGIS.

Scale Bar
Displays a scale bar on the map canvas. You can customize the placement, style, color, and size of the scale bar.

WFS
Experimental plugin for consuming WFS services on the Internet and displaying the

data in QGIS. This plugin is included as part of the QGIS package, but at the time of this writing probably isn't ready for prime time.

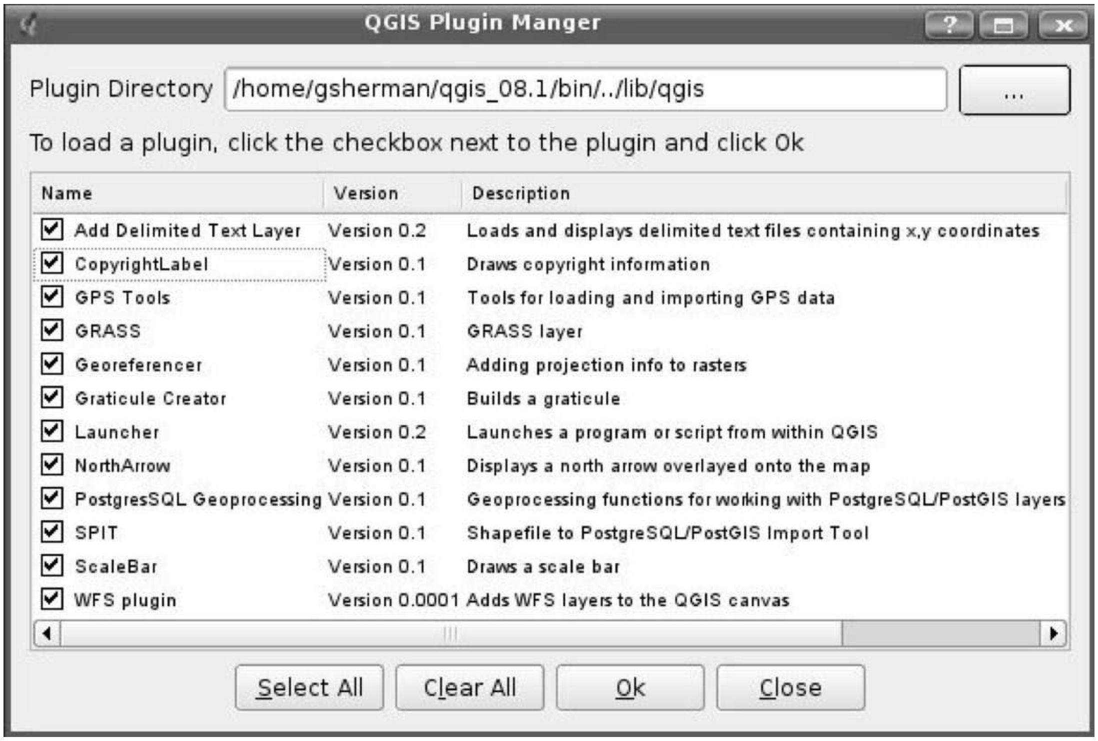
4.1 The Plugin Manager

The Plugin Manager allows you to load and unload QGIS plugins. Before you can use a plugin, it must be loaded. By default, no plugins are loaded the first time you run QGIS. Once you have loaded a plugin, QGIS remembers and will load it the next time you start it up.

When you load a plugin, new tools and menu items are added to the user interface. The Plugin Manager is shown below, with all plugins loaded.

Tip 5 Using Plugins

Most plugins add both a tool to the Plugins toolbar and an entry under the main Plugins menu. The menu items are organized by sub-menus, since some plugins or plugin categories have multiple tools. To use a plugin, you can either click on the tool in the toolbar or use the menu.



Plugin Directory /home/gsherman/qgis_08.1/bin/../lib/qgis

To load a plugin, click the checkbox next to the plugin and click Ok

Name	Version	Description
☑ Add Delimited Text Layer	Version 0.2	Loads and displays delimited text files containing x,y coordinates
☑ CopyrightLabel	Version 0.1	Draws copyright information
☑ GPS Tools	Version 0.1	Tools for loading and importing GPS data
☑ GRASS	Version 0.1	GRASS layer
☑ Georeferencer	Version 0.1	Adding projection info to rasters
☑ Graticule Creator	Version 0.1	Builds a graticule
☑ Launcher	Version 0.2	Launches a program or script from within QGIS
☑ NorthArrow	Version 0.1	Displays a north arrow overlayed onto the map
☑ PostgresSQL Geoprocessing	Version 0.1	Geoprocessing functions for working with PostgreSQL/PostGIS layers
☑ SPIT	Version 0.1	Shapefile to PostgreSQL/PostGIS Import Tool
☑ ScaleBar	Version 0.1	Draws a scale bar
☑ WFS plugin	Version 0.0001	Adds WFS layers to the QGIS canvas

Let's take a look at some of the key plugins available in QGIS.

4.2 The Delimited Text Plugin

The delimited text plugin allows you to treat a text file as a layer in QGIS. The text file must have the following characteristics:

1. Plain text format
2. Delimited with any character

3. A header row with the field names
4. An X and Y field (can be in any coordinate system)

The plugin parses your text file and adds it to the map canvas as a layer. If you like, you can save it to a shapefile by right-clicking on the layer name in the legend and choosing Save as shapefile... from the popup menu.

Let's look at an example. We have been lucky enough to download some historical earthquake data and manipulate it using a bit of Ruby (that's another story) to create a delimited text file in the following format:

```
event_date|event_time|latitude|longitude|depth|magnitude
06/29/1898|18:36:00.0|52.0000|172.0000|0.0|7.6|
10/11/1898|16:37:32.7|50.7100|-179.5|0.0|6.9|
07/14/1899|13:32:00.0|60.0000|-150.0|0.0|7.2|
09/04/1899|00:22:00.0|60.0000|-142.0|25.0|7.9|
09/04/1899|04:40:00.0|60.0000|-142.0|0.0|6.9|
09/10/1899|17:04:00.0|60.0000|-140.0|25.0|7.4|
09/10/1899|21:41:00.0|60.0000|-140.0|25.0|8.0|
09/17/1899|12:50:00.0|59.0000|-136.0|0.0|6.9|
09/23/1899|11:04:00.0|60.0000|-143.0|0.0|6.9|
...
11/29/2006|23:00:57.1|51.0573|-179.4543|13.9|4.2|
```

The first row is the header and contains the names of the fields in the file. The delimiter is "|", but we could have just as easily used "," or some other character.

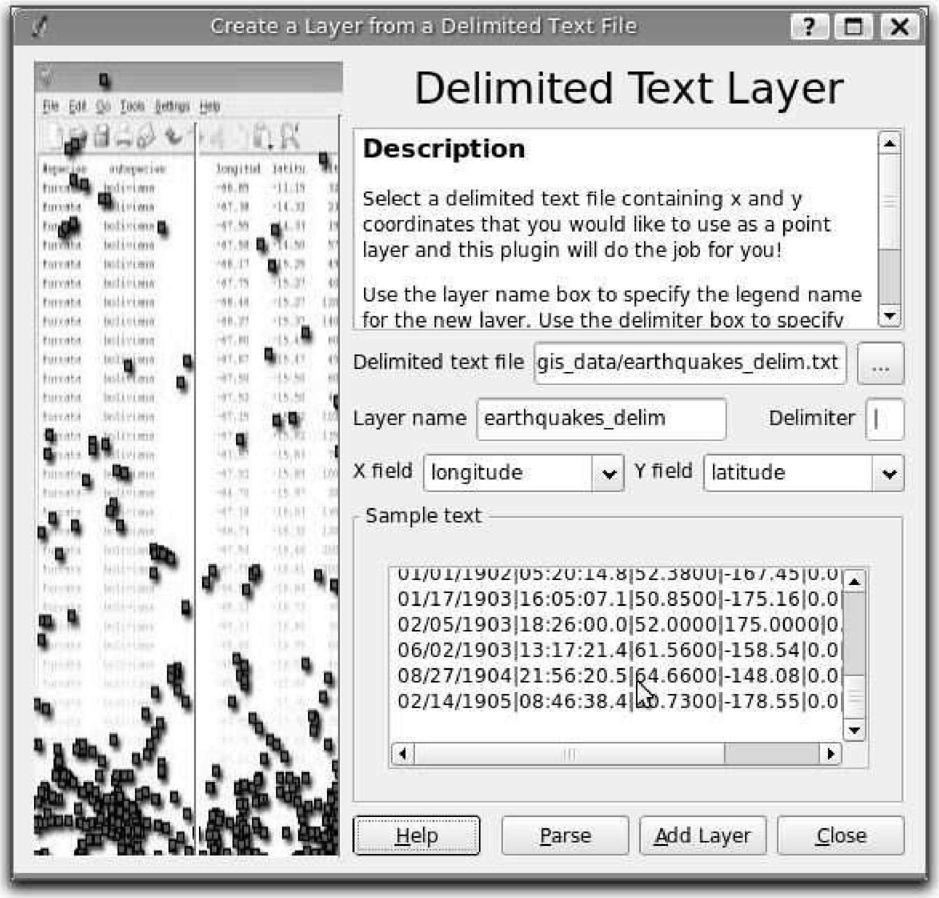
Exercise 11 Create a Delimited Text Layer

To create a layer:

1. Open the Plugin Manager
2. Enable the "Add Delimited Text" plugin
3. Click OK
4. Find the Add Delimited Text Layer tool in the Plugins toolbar and click on it to bring up the plugin dialog
5. To specify the text file, click on the ... button, browse to the data directory, and choose `earthquakes_delim.txt`
6. Click Open
7. If the delimiter isn't already |, set it now and click Parse to parse the text file again
8. Enter a name for the layer: Earthquakes
9. From the drop-down boxes, choose the field names for the X and Y values
10. Click Add Layer to add it to the map canvas
11. Close the dialog

Once the layer is added to the map canvas, it behaves just like any other layer. You can zoom in and out, identify features, and open the attribute table. If you want to use the layer in the future, you should save it to a shapefile.

Our completed dialog used to add the layer looks like this:



Description

Select a delimited text file containing x and y coordinates that you would like to use as a point layer and this plugin will do the job for you!

Use the layer name box to specify the legend name for the new layer. Use the delimiter box to specifv

Delimited text file: gis_data/earthquakes_delim.txt

Layer name: earthquakes_delim

Delimiter: |

X field: longitude

Y field: latitude

Sample text

```
01/01/1902|05:20:14.8|52.3800|-167.45|0.0
01/17/1903|16:05:07.1|50.8500|-175.16|0.0
02/05/1903|18:26:00.0|52.0000|175.0000|0.
06/02/1903|13:17:21.4|61.5600|-158.54|0.0
08/27/1904|21:56:20.5|64.6600|-148.08|0.0
02/14/1905|08:46:38.4|[illegible]0.7300|-178.55|0.0
```

4.3 The Decoration Plugins

The "Decoration" plugins include the following:

1. Copyright Label
2. North Arrow
3. Scale Bar

These "decorate" the map by adding cartographic elements.

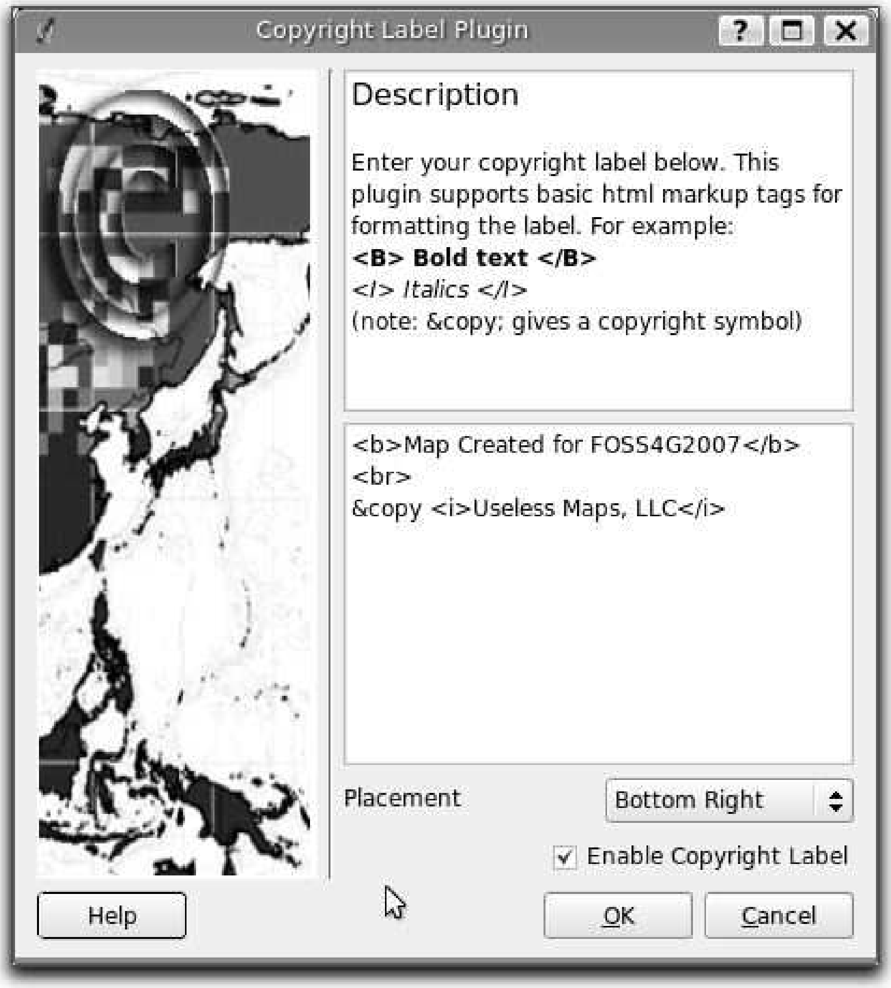
4.3.1 Copyright Label

⋆⋆**Try It**

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

1. Make sure the plugin is loaded
2. Click on the (Copyright Label) tool on the Plugins 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 drop-down box
5. Make sure the "Enable Copyright Label" checkbox is checked

6. Click OK



Description

Enter your copyright label below. This plugin supports basic html markup tags for formatting the label. For example:
<B> Bold text </B>
<I> Italics </I>
(note: &copy; gives a copyright symbol)

```
<b>Map Created for FOSS4G2007</b>
<br>
&copy <i>Useless Maps, LLC</i>
```

Placement: Bottom Right

☑ Enable Copyright Label

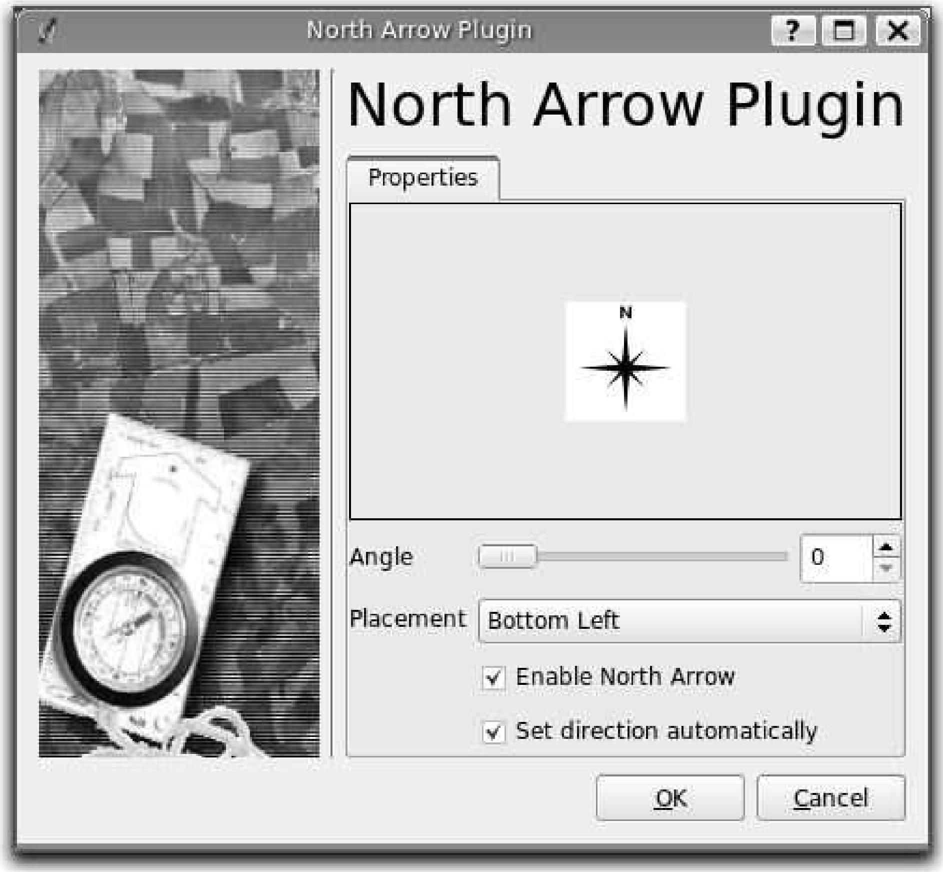
In the example above, the first line is in bold, the second (created using <br>) contains a copyright symbol, followed by our company name in italics.

4.3.2 North Arrow

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.

⋆⋆**Try It**

For placement of the arrow you have four options, corresponding to the four corners of the map canvas.



Angle 0

Placement Bottom Left

☑ Enable North Arrow

☑ Set direction automatically

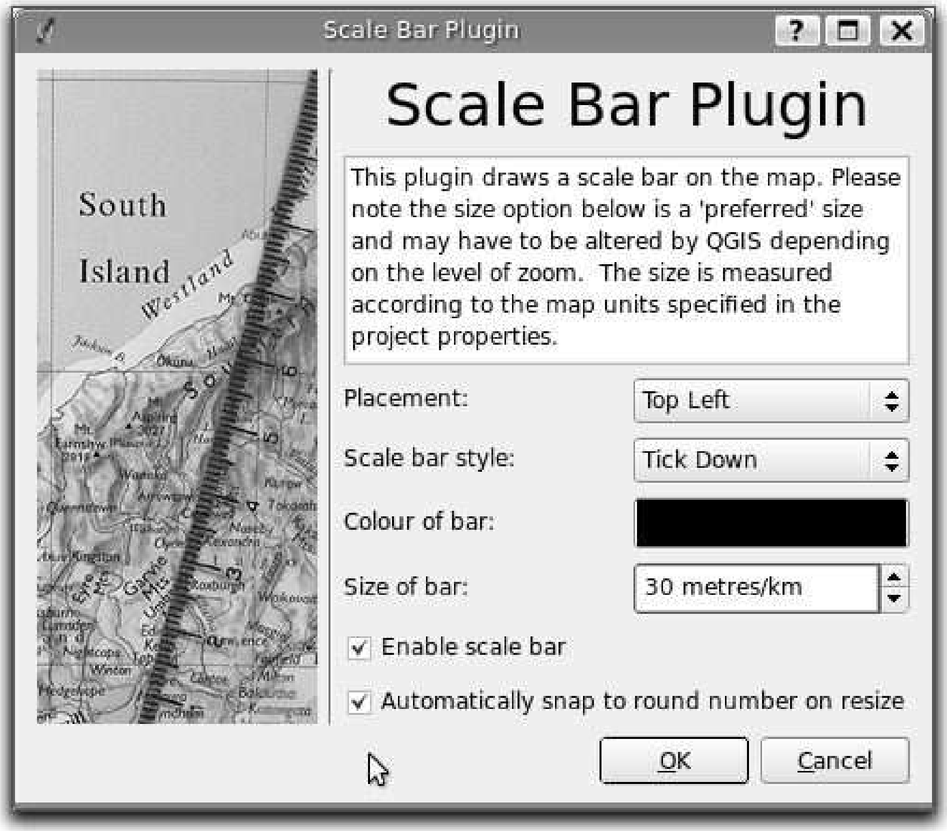
4.3.3 Scale Bar

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.

⋆⋆Try It To add a scale bar:

1. Open the plugin dialog by clicking on the (Scale Bar) tool on the Plugins toolbar
2. Choose the placement from the drop-down list
3. Choose the style
4. Select the color for the bar, or use the default black color
5. Set the size of the bar and its label
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
8. Click [OK]



Scale Bar Plugin

This plugin draws a scale bar on the map. Please note the size option below is a 'preferred' size and may have to be altered by QGIS depending on the level of zoom. The size is measured according to the map units specified in the project properties.

Placement: Top Left

Scale bar style: Tick Down

Colour of bar: ■

Size of bar: 30 metres/km

☑ Enable scale bar

☑ Automatically snap to round number on resize

4.4 The GPS Plugin

The GPS plugin allows you to upload and download data as well as open existing GPX files.

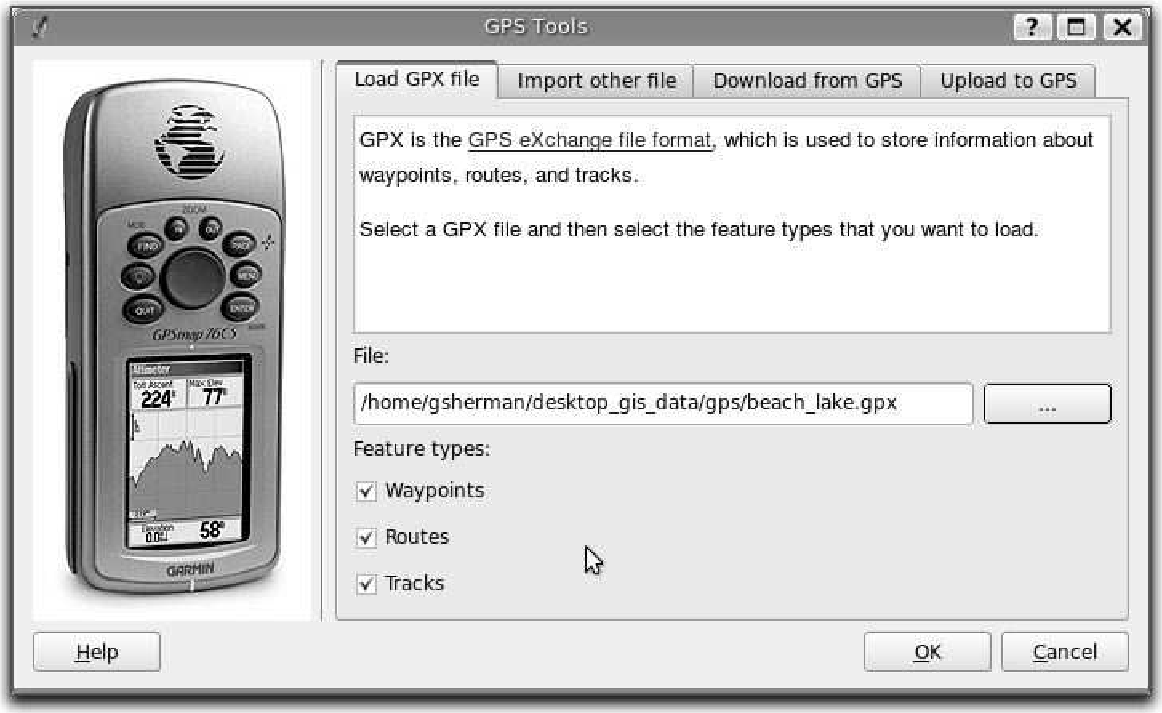
QGIS uses gpsbabel to import other formats and for GPS downloading and uploading operations. Fortunately gpsbabel runs on the same platforms as QGIS so if you can run QGIS, you can use it with your GPS. It supports 100+ formats and a good selection of GPS hardware. For upload/download, if you have a Garmin^{TM}or Magellan^{TM}unit, you should be good to go. For others, see the format list on the web page to see if yours is listed there.[1]

When you load the plugin, the first thing to notice is the four tabs across the top of the GPS plugin dialog. The first two (”Load GPX file“ and “Import other file”) allow you to load data stored on a disk, CD, or maybe a thumb drive. The last two tabs (”Download from GPS” and “Upload to GPS”) provide the tools needed to move data to and from your unit.

Using the functions available on the “Import other file” tab you can convert and import any format supported by gpsbabel.

More information on using the GPS plugin can be found in the *QGIS User Guide*.

[1] http://www.gpsbabel.org/



GPX is the GPS eXchange file format, which is used to store information about waypoints, routes, and tracks.

Select a GPX file and then select the feature types that you want to load.

File: /home/gsherman/desktop_gis_data/gps/beach_lake.gpx

Feature types:

☑ Waypoints

☑ Routes

☑ Tracks

4.5 Graticule Creator

The graticule creator lets us create a "grid" of points, line, 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.

Exercise 12 Creating a Graticule

To create a graticule:

1. Make sure the plugin is loaded
2. Click on the (Graticule Creator) tool on the plugins toolbar
3. Choose the type of graticule you which to create: point, line, or polygon
4. Enter the latitude and longitude for the lower left and upper right corners of the graticule
5. Enter the interval to be used in constructing the grid. You can enter different values for the X and Y directions (longitude, latitude)
6. Choose the name and location of the shapefile to be created
7. Click OK to create the graticule and add it to the map canvas

4.6 Other Plugins

We'll look at some of the other plugins later. For those we don't cover, feel free to explore them on your own.

Chapter 5

Making MapServer Map Files

QGIS can be used to create map files for MapServer. You use QGIS to "compose" your map by adding and arranging layers, symbolizing them, and customizing the colors.

In order to use the MapServer exporter, you must have Python on your system and QGIS must have been compiled with support for it.

5.1 Creating the Project File

To create a MapServer map file:

1. Add your layers to QGIS
2. Symbolize your layers, setting the renderer and colors
3. Arrange the layers in the order you want them to appear in MapServer
4. Save your work to a QGIS project file

This gets us to the point where we are ready to create the map file.

Tip 6 MAPSERVER EXPORT REQUIRES A QGIS PROJECT FILE

This has been a source of confusion for a number of people. The MapServer export tool operates on a saved QGIS project file, **not** the current contents of the map canvas and legend. When using the tool, you need to specify a QGIS project file as input.

5.2 Creating the Map File

The exporter tool (`msexport`) is installed in your QGIS binary directory and can be used independently of QGIS.

From QGIS you can start the exporter by choosing Export to MapServer Map... from the File menu.

Here is a summary of the input fields:

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

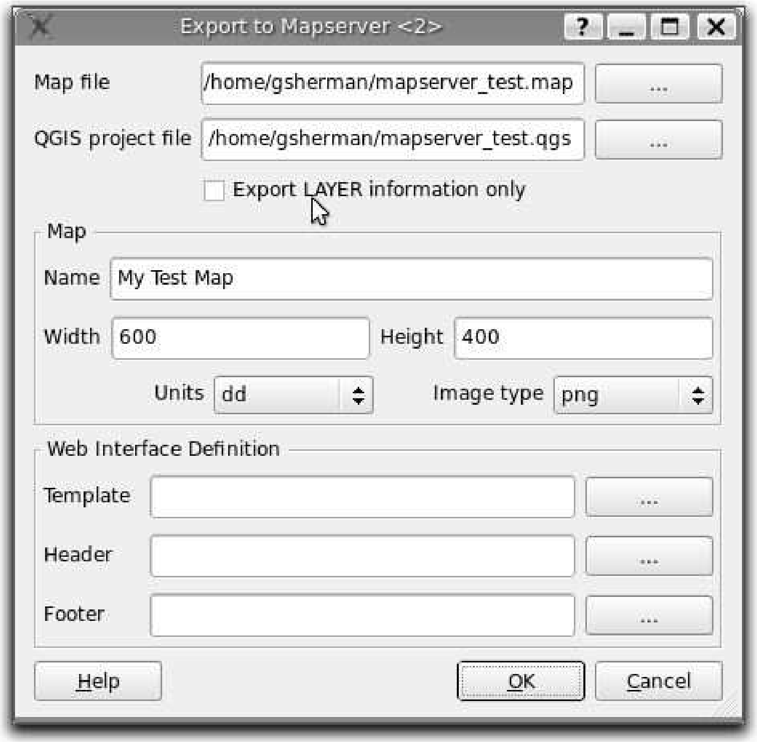
Web Footer
Full path to the MapServer footer file to be used with the map file

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 it's still way better than writing a map file from scratch.

Exercise 13 Creating a Map File

Let's create a map file using our `world_borders` and `cities` layers:

1. Load the `world_borders` and `cities` layers into QGIS
2. Change the colors and symbolize the data as you like
3. Save the project using Save Project from the File menu
4. Open the exporter by clicking on Export to MapServer Map... in the File menu
5. Enter a name for your new map file
6. Browse and find the project file you just saved
7. Enter a name for the map
8. Enter 600 for the width and 400 for the height
9. Our layers are in decimal degrees so we don't need to change the units
10. Choose "png" for the image type
11. Click OK to generate the map file



Export to Mapserver <2>

Map file: /home/gsherman/mapserver_test.map

QGIS project file: /home/gsherman/mapserver_test.qgs

□ Export LAYER information only

Map

Name: My Test Map

Width: 600 Height: 400

Units: dd Image type: png

Web Interface Definition

Template:

Header:

Footer:

You'll notice there is no feedback on the success of your efforts. This is an enhancement scheduled for the next version.

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

5.3 Testing the Map File

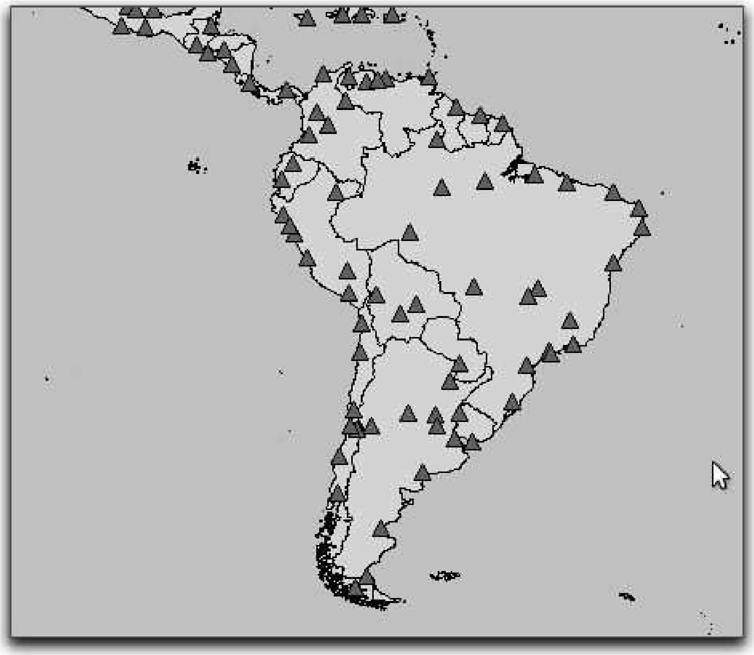
Let's test our work by using the `shp2img` command to create an image from the map file. The `shp2img` utility is part of MapServer, but is also distributed with FWTools. To create an image from our map:

1. Open a terminal window
2. If you didn't save your map file in your home directory, change to the directory where you saved it
3. Run shp2img
4. View the created image

Assuming our map file was named `mapserver_test.map`, the shp2img command is:

```
shp2img -m mapserver_test.map -o mapserver_test.png
```

This creates a PNG for us to view, containing all the layers that were on when we saved the QGIS project. In addition, the extent of the PNG will be the same as when we saved the project.



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.[1]

[1] `http://openlayers.org/presentations/mappingyourdata/`

Chapter 6

Consuming WMS and WFS Data

QGIS supports WMS and WFS as data sources. The support for WFS is preliminary at this time. WMS support is native; WFS is implemented using a plugin.

6.1 Using WMS Data

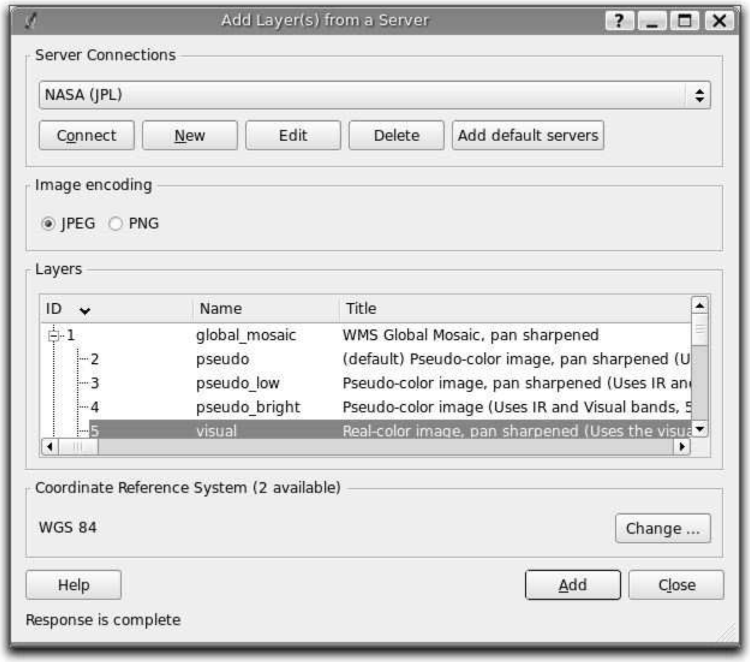
To add WMS data to your map, click on the (Add WMS Layer) tool to bring up the dialog or you can choose Add WMS Layer... from the Layer menu.

The first time you use the WMS feature, there are no servers defined. 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.

Exercise 14 Loading a WMS Layer

To add a layer from the server:

1. Click on the (Add WMS Layer) tool
2. If you haven't already done it, click on the [Add default servers] button
3. Select "NASA (JPL)" from the drop-down list
4. Click [Connect]
5. The "Layers" list will be populated with the available layers
6. Click on layer id 1 to expand it and see the options available
7. Choose one of the options by clicking on it
8. Click [Add] to add it to the map



NASA (JPL)

● JPEG ○ PNG

ID	Name	Title
1	global_mosaic	WMS Global Mosaic, pan sharpened
2	pseudo	(default) Pseudo-color image, pan sharpened (U
3	pseudo_low	Pseudo-color image, pan sharpened (Uses IR an
4	pseudo_bright	Pseudo-color image (Uses IR and Visual bands, 5
5	visual	Real-color image, pan sharpened (Uses the visua

WGS 84

Response is complete

You can add additional layers by repeating the process.

6.1.1 Adding a New WMS Connection

⋆⋆**Try It**

You probably will want to use more WMS servers than the default set added by QGIS. First you need to find the URL for the server. Once you have the URL, create a connection. Let's create a connection to the QGIS WMS server:

1. Click on the (Add WMS Layer) tool
2. Click the [New] button
3. Give the connection a name – we'll use "QGIS WMS"
4. Enter the URL – `http://qgis.org/cgi-bin/mapserv?map=/var/www/maps/main.map`
5. If a proxy is required, enter the information – for QGIS we don't need one
6. Click [OK] to create the connection

You can now select the connection from the drop-down list and try it out. Try loading the "Users" layer to the map.

You can identify features on the WMS layer, depending on the capabilities implemented by the owner.

6.1.2 Setting Options

Depending on the service, you can change the image format and the coordinate system when loading a layer. First open the connection and click on the layer you want to load. Then set the image type and projection from the available options.

6.2 Using WFS Data

In QGIS, a WFS layer behaves pretty much like any other vector layer. You can identify and select features and view the attribute table. The WFS plugin doesn't support editing at this time.

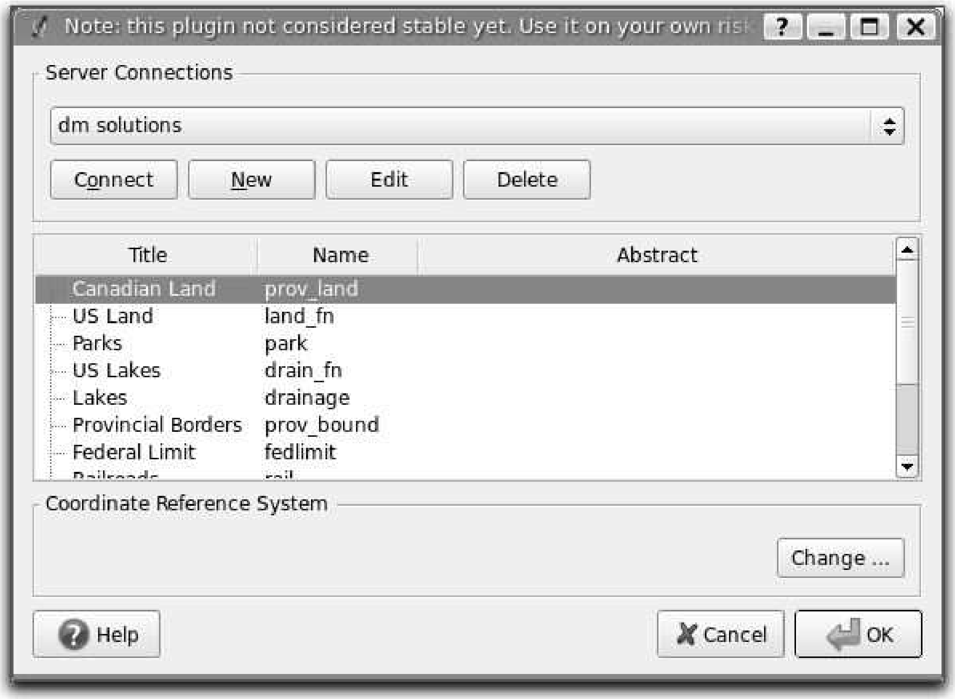
Adding a WFS layer is very similar to the procedure used with WMS. The difference is there are no default servers defined, so we have to add our own.

Let's use the DM Solutions WFS server. The URL is:
`http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?VERSION=1.0.0&SERVICE=wfs&REQUEST=GetCapabilities`

Exercise 15 Loading a WFS Layer

To add the server and display a layer:

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 "DM Solutions" as the name
5. Enter the URL (see previous page)
6. Click [OK]
7. Choose "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



Note: this plugin not considered stable yet. Use it on your own risk

Server Connections: dm solutions

Title	Name	Abstract
Canadian Land	prov_land	
US Land	land_fn	
Parks	park	
US Lakes	drain_fn	
Lakes	drainage	
Provincial Borders	prov_bound	
Federal Limit	fedlimit	
Railroads	rail	

You'll notice there is no feedback while connecting to the server or while loading a layer.

Once the layer is loaded, you can identify and select a province or two and view the attribute table.

⋆⋆**Try It**

Remember this plugin is experimental—you might also experience random behavior and crashes. You can look forward to improvements in a future version of the plugin.

Tip 7 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, some of them maintained and some not, that list public servers you can use.

Chapter 7

Viewing and Editing Data in PostGIS

QGIS has had support for PostGIS from day one. It was the first data store supported in version 0.0.1. Since then the support has evolved to include editing, view support, and subset layers.

7.1 Viewing Data

Support for PostGIS is part of core QGIS—you don't have to load a plugin to access it. Before you can access PostGIS data, you obviously need a PostgreSQL database with PostGIS. We won't go into the details of the installation process, but rather assume we have a properly configured database ready for our use.

7.1.1 Creating a Connection

In order to get at the data in our database, we need to create a connection. This requires some information about the database server, the database, and your login information.

Exercise 16 Creating a PostGIS Connection

To create a new connection:

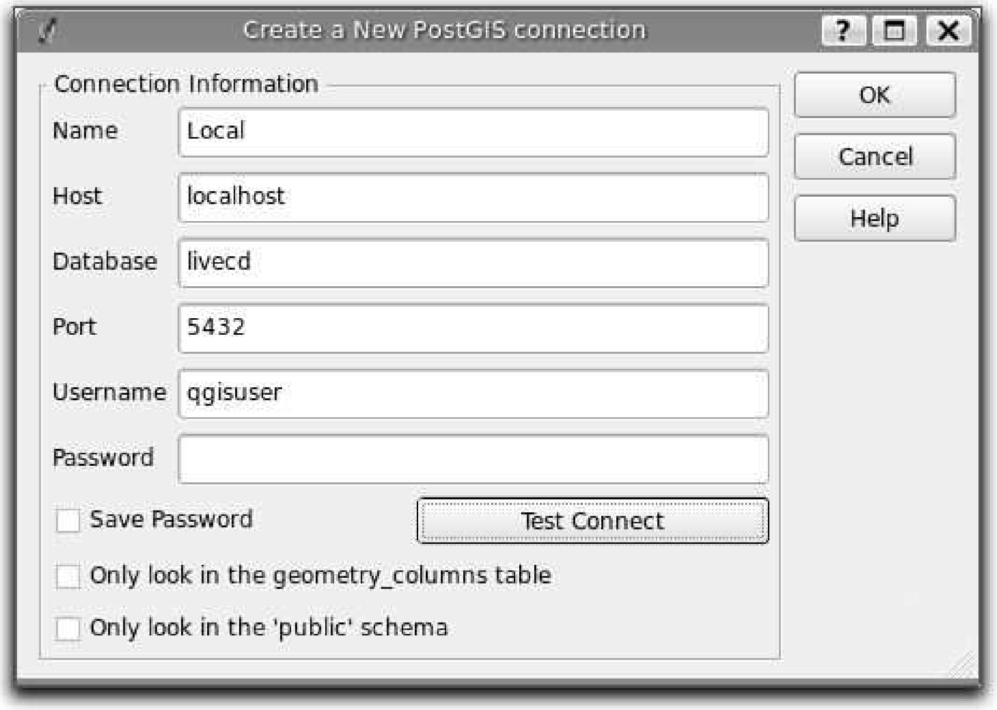
1. Click on (Add a PostGIS Layer) in the toolbar
2. Click [New]
3. Enter a name for the connection—this can be anything handy to remember
4. Enter the host name. If your PostGIS database is on the same machine, this will be *localhost*
5. Enter the name of the database that houses your data: livecd
6. Enter the port. The default PostgreSQL port is 5432.
7. Enter the user name: qgisuser
8. Enter the password (you may not need one if running QGIS on the same machine as PostgreSQL)
9. Click [Test Connect] to see if you got things correct
10. Click [OK] to save the connection

If access to your database requires a password you can have QGIS store it for you, however this may be a security risk depending on your environment.

QGIS searches all the tables in your database looking for those that have a spatial column. You can configure the connection to only look for layers in the **geometry_columns** table. This will speed up the populating of the layer list, but will miss layers that aren't in the **geometry_columns** table. If you always create your PostGIS layers in a way that creates an entry in the **geometry_columns** table, then it's safe to check this option.

The other option allows you to only search the "public" schema for spatial tables. This also speeds up the populating of the layer list but is only helpful if you layers are always stored in the "public" schema.

Our completed connection dialog is shown below:



Name: Local
Host: localhost
Database: livecd
Port: 5432
Username: qgisuser
Password:

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

7.1.2 Loading Data

Let's use the new connection to load a layer.

Exercise 17 Loading the Cities Layer from PostGIS

To load the `cities` layer from PostGIS:

1. Click on Add a PostGIS Layer in the toolbar
2. Choose your connection from the drop-down box
3. Click Connect
4. Choose the `cities` layer from the list
5. Change the encoding if you need to
6. Click Add

Once loaded, the PostGIS layer behaves like any other vector layer.

7.1.3 Creating Subsets

When loading a PostGIS layer, you can specify a query to limit the features that are returned. For example, say we only want to display cities with populations of 4 million or more. We can do this when we add the cities layer, effectively creating a new or "subset" layer:

Exercise 18 Creating a Subset from the Cities Layer

1. Click on Add a PostGIS Layer in the toolbar
2. Choose your connection from the drop-down box
3. Click Connect
4. Double-click on the **cities** layer to bring up the query builder
5. Double-click the “population” field to add it to the where clause
6. Click the >= button
7. In the where clause box, type 4000000
8. Click Test to test the query and see how many features match
9. Click OK. The query now shows up in the “Sql” column of the `cities` layer in the layer list.
10. Click Add to add our subset layer to the map

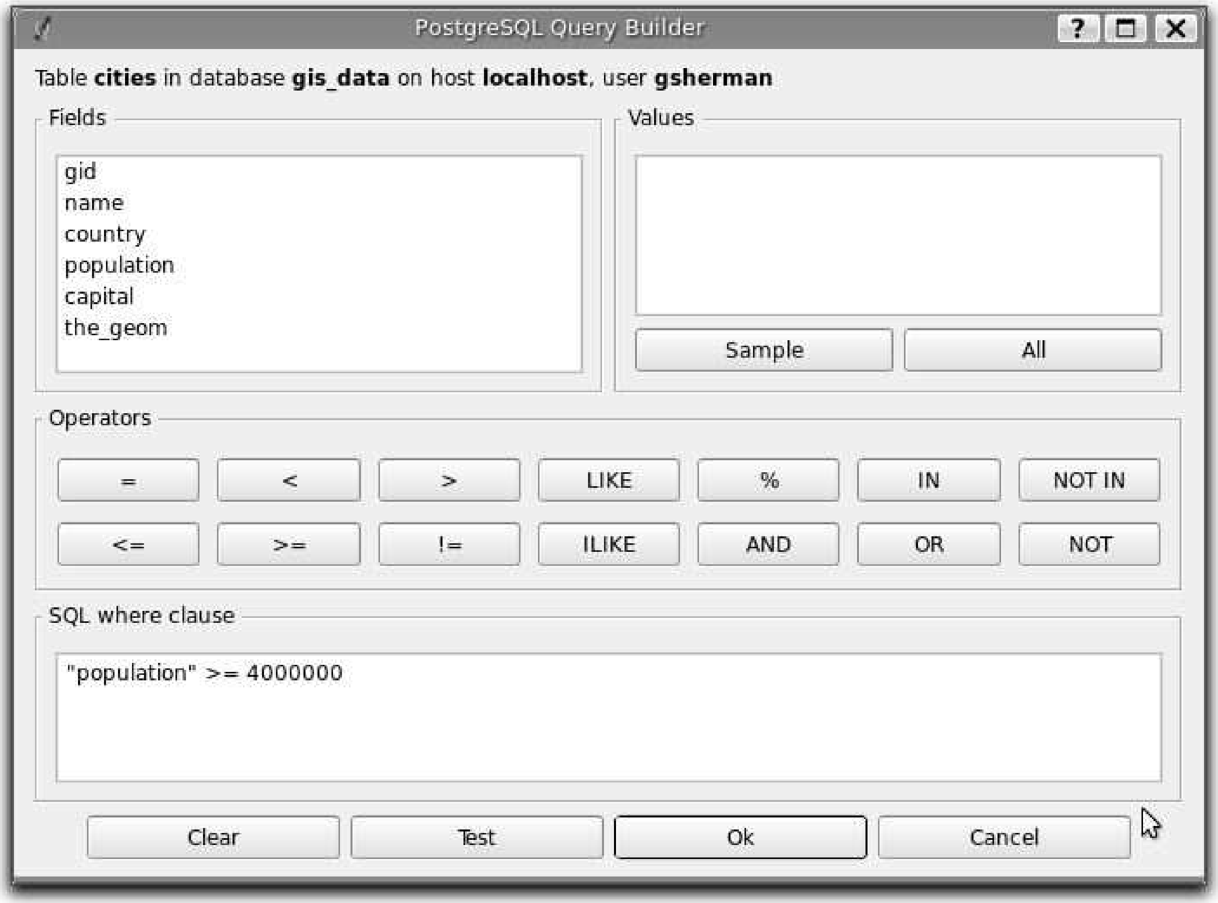


Table **cities** in database **gis_data** on host **localhost**, user **gsherman**

Fields: gid, name, country, population, capital, the_geom

SQL where clause: "population" >= 4000000

Modifying the Subset

⋆⋆**Try It** Once you have a subset layer loaded, you can tweak the query to modify it without dropping it from the map and adding it again. In fact, you can do this with any PostGIS layer:

1. Open the vector properties dialog by double-clicking on the PostGIS layer in the legend

or right-clicking and choosing Properties from the popup menu

2. Click on the "General" tab
3. Click on Query Builder at the bottom right of the dialog
4. Edit the query using the same methods we used to create the original subset layer
5. Click OK to apply the query
6. Close the properties dialog

7.1.4 Creating a View

With a bit of SQL, you can create a view. Think of it as a permanent subset (or superset) of your layer.

To make our layer of cities with greater than 4 million people more permanent, we create a view using `psql` or PgAdminIII:

```
create view big_cities as select * from cities where population > 4000000;
```

The `big_cities` layer now shows up in our list when we go to add a PostGIS layer. When we add it to the map, it contains only those cities matching our criteria.

QGIS found the view automatically. If you were to look in the **geometry_columns** table you would see there is no entry for it. If you checked the box to "Only look in the geometry_columns table" when you created your PostGIS connection, you won't see your view in the layer list.

7.2 Editing Data

Editing PostGIS data is done exactly like we described in Section 3.1 on page 21. All the techniques described previously work with PostGIS layers. There are a couple of things we can do with PostGIS layers that can't be done with other types when it comes to editing attributes.

You may have noticed when we were editing attributes back in 2.5.2 on page 17 that a couple of the buttons on the attribute table dialog were disabled. These are the buttons to add or delete a column. While these don't work for a shapefile, they do for a PostGIS layer. For example, to add a column to the attribute table for the `cities` layer:

1. Make sure the `cities` layer is loaded
2. Make the layer active by selecting it in the legend (just click on its name)
3. Open the attribute table by clicking on the Open Table tool
4. Click Start editing
5. Click the New column button (the one with the yellow starburst)
6. Let's add a "Visited" column so we can flag which cities we've been to
7. Enter "visited" as the column name

8. Choose varchar(30) as the type

9. Click OK

We now have a new column in the table. We can enter values into the column using the editing capabilities of the attribute table, or we could do a bulk update using a SQL query from **psql**, the PostgreSQL interactive terminal.

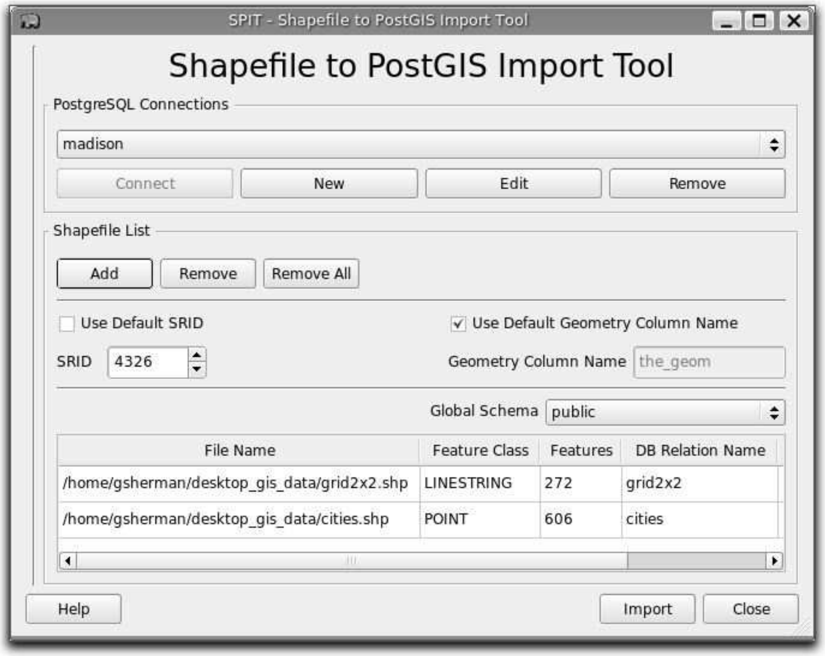
This provides a convenient way to create a new column. To commit the changes, click the Stop editing button and save the changes.

Likewise you can delete a column. Start editing the attribute table, then click the Delete column button and select the column to delete from the popup dialog.

7.3 Loading Data

Thus far we assumed you already had data in PostGIS. QGIS provides a way to load shapefiles into PostGIS using the SPIT plugin.

To load shapefiles, add them to the list using the Add button. You can change the SRID, name of the geometry column, the schema to load it into, and the name of the database table to create if you aren't happy with the default.


SPIT - Shapefile to PostGIS Import Tool
Shapefile to PostGIS Import Tool
Connect
New
Edit
Remove
Shapefile List
Add
Remove
Remove All
Help
Import
Close

PostgreSQL Connections: madison

☐ Use Default SRID

SRID 4326

☑ Use Default Geometry Column Name

Geometry Column Name the_geom

Global Schema public

File Name	Feature Class	Features	DB Relation Name
/home/gsherman/desktop_gis_data/grid2x2.shp	LINESTRING	272	grid2x2
/home/gsherman/desktop_gis_data/cities.shp	POINT	606	cities

When you click Import each of the shapefiles in the list will be imported into PostGIS.

Tip 8 PostGIS Import Problems

You may encounter problems using SPIT to import your data into PostGIS. SPIT isn't particularly adept at handling errors or irregularities in your data. Fortunately PostGIS comes with the `shp2pgsql` command line utility to load shapefiles. If you encounter trouble with SPIT, try using `shp2pgsql` to do the job.

Chapter 8

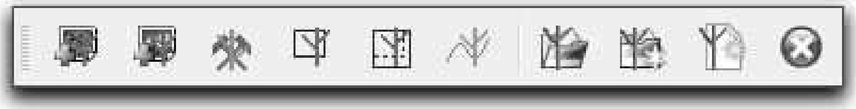
GRASS and QGIS

The GRASS plugin provides access to GRASS from within QGIS. This includes the ability to view, edit, and create data, as well as perform analysis using the GRASS geoprocessing modules.

In this chapter we'll introduce the plugin and some of the ways you can use it to work with GRASS data.

8.1 The GRASS Plugin

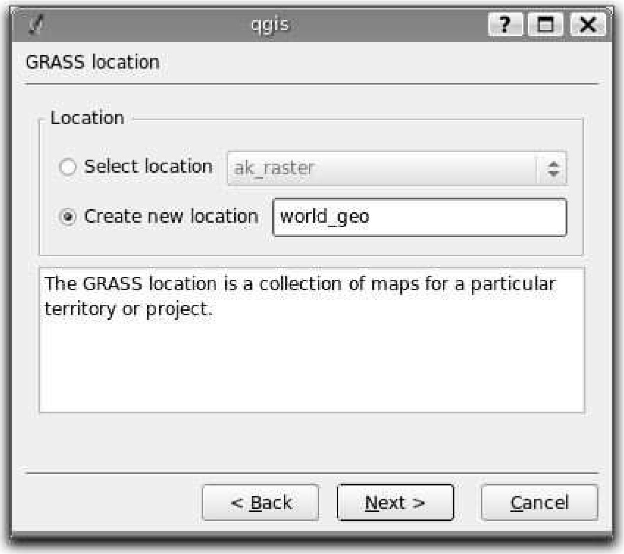
The GRASS plugin is loaded from the Plugin manager just like all QGIS plugins. When you load it, a new toolbar will appear on the user interface.[1]



Normally in GRASS you open a mapset first, then you can load a map (layer). With the QGIS-GRASS plugin loaded, we can open any GRASS layer we like without opening a mapset—but first we must get some data into GRASS.

8.2 Creating a Location

GRASS stores data in a "location" which represents a specific area with a specific coordinate system. In order to use GRASS data, we must import it into a `location`.[2]



○ Select location: ak_raster

● Create new location: world_geo

The GRASS location is a collection of maps for a particular territory or project.

[1] The GRASS plugin is unique in that it creates its own toolbar

[2] This is not strictly true—you can view external data sets without importing them

Exercise 19 Creating a GRASS Location Using QGIS

Let's create a location for our world wide data:

1. Start QGIS
2. Make sure the GRASS plugin is loaded
3. Load the `world_borders` shapefile
4. In the GRASS toolbar, click on the (New mapset) tool to bring up the mapset wizard
5. Each location is stored in a directory. Select an existing data directory or create a new one for storing the location
6. Click [Next]
7. We can use this wizard to create a new mapset within an existing location or create a new location altogether. Click "Create new location" radio button
8. Enter a name for the location—we'll use world_geo
9. Click [Next]
10. Define the projection by clicking on the "Projection" radio button to enable the projection list
11. We are using WGS 84 geographic. Since we happen to know that its SRID is 4326, we enter it in the search box
12. Click [Find] to select the projection
13. Click [Next]
14. To define the default region, enter the bounds in the north, south, east, and west direction. Fortunately these are already correct for our world geographic location (-180,-90, 180, 90)
15. Click [Next]
16. We need to define a mapset within our new location. Name it whatever you like—your username is a good choice
17. Check out the summary to make sure it's correct
18. Click [Finish]
19. The mapset and location are created and opened as the current working set
20. Notice that some of the tools in the GRASS toolbar that were disabled are now enabled for us to use

If that seemed like a lot of steps, it's really not all that bad and a very quick way to create a location.

Our location is now ready for use. To view the default region, click the (Display Current Grass Region) tool.

Now we are ready to import some data into GRASS.

8.3 Importing Data

To import data, you could use, among others, the `v.in.ogr` or `r.in.gdal` modules from the GRASS command line. We'll use the QGIS-GRASS plugin to do the work for us. One thing to remember when using the plugin to import data into GRASS—you must load the layer you want to import into QGIS, then use the import module.

8.3.1 Importing Vectors

Let's start out by importing our `cities` shapefile into GRASS:

Exercise 20 Importing the Cities Shapefile into GRASS

1. Add the `cities` shapefile to QGIS
2. If it's not already open, open the mapset we just created using the (Open mapset) tool on the GRASS toolbar
3. Open the GRASS toolbox by clicking on the (Open GRASS tools) tool
4. Under "Import", locate `v.in.ogr` and click on it to bring up the OGR import module
5. From the drop-down list, select the `cities` layer
6. Name the layer "cities"
7. Click [Run] to start the import
8. Review the results on the "Output" tab
9. Click [View output] to add the new GRASS layer to the map canvas
10. Close the GRASS toolbox and examine the new layer

You might have noticed on the "Output" tab that it not only shows the processing, but the actual GRASS command used to perform the operation. This is a good way to learn and understand how the GRASS modules are used.

The new `cities` layer behaves like any other vector layer in QGIS. You can open the attribute table and identify features. If you like, import the `world_borders` shapefile into GRASS.

It will take about 6 minutes to import the world borders

8.3.2 Importing Rasters

The process for importing a raster is pretty much the same as for a vector layer. First add it to the map, then choose `r.in.gdal` module to do the import.

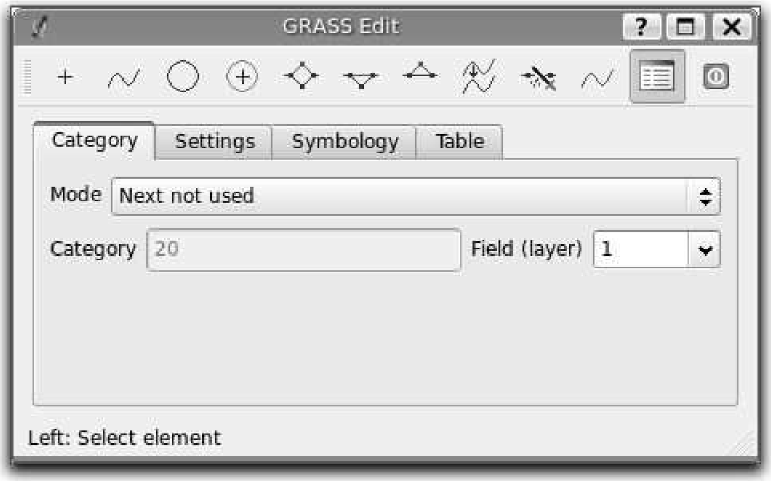
Let's import the DEM into our location:

Exercise 21 Importing a Raster (DEM) into GRASS

1. Add the `ancc6_geo.DEM` to QGIS
2. Make sure your mapset is open
3. Open the GRASS toolbox
4. If it's not active, click on the "Modules" tab
5. Find the `r.in.gdal` module in the toolbox and click on it
6. Choose `ancc6_geo` from the drop-down list
7. Enter `ancc6_geo` for the name of the output raster
8. Click [Run]
9. Click [View output] to add it to the map
10. Close the GRASS toolbox
11. Examine the new raster

8.4 Digitizing and Editing

Editing of GRASS layers is done using the plugin. You can edit both the features and their attributes using the (Edit Grass Vector Layer) tool. To edit a layer, activate it in the legend and then click the tool. This brings up the GRASS Edit dialog:



GRASS Edit

Category | Settings | Symbology | Table

Mode: Next not used

Category: 20 Field (layer): 1

Left: Select element

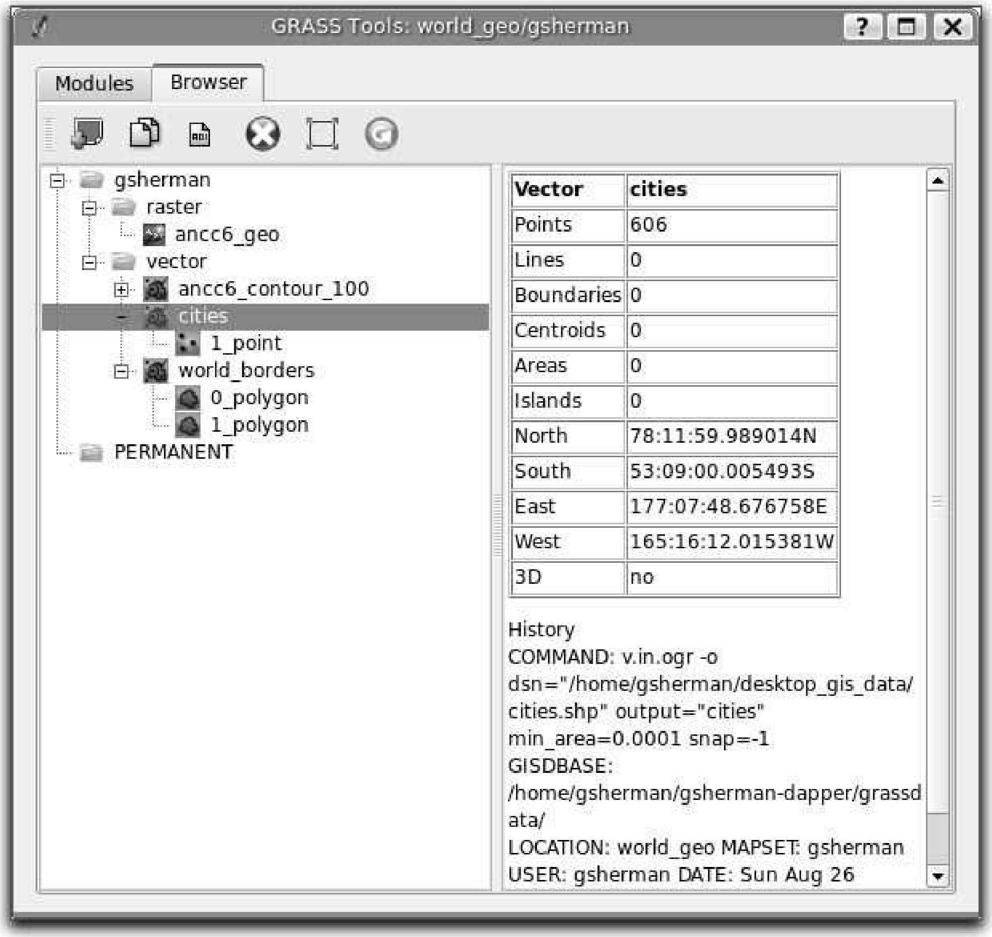
There are tools for creating the various feature types and editing the attribute table. We won't go into detail about editing with GRASS—see the QGIS user manual for more details on the GRASS vector model and digitizing/editing with QGIS.

With QGIS 0.8 and higher you can create an empty GRASS vector layer to use for digitizing features. To create a new layer, click the (Create new Grass Vector) tool on the GRASS toolbar and enter a name for the new layer. When you click [OK], the layer is created and the Grass Edit dialog opens, ready for you to begin creating features. By default the new layer contains only one field but you can add others using the tools on the "Table" tab.

8.5 Using the Browser

The GRASS browser allows you to perform a number of operations on your layers, as well as get information about each. To activate the browser, open the Grass Toolbox and click on the "Browser" tab. All of the mapsets in the current location are displayed on the left in a tree structure. Typically you will see the PERMANENT mapset, along with the one or more of your user mapsets.

If a mapset contains maps, you will be able to expand the tree. The maps are further categorized into raster, region, and vector nodes. Expanding one of the nodes will show you a list of all the maps associated with it. If you click on a map, the pane on the right displays information about the map.



Vector	cities
Points	606
Lines	0
Boundaries	0
Centroids	0
Areas	0
Islands	0
North	78:11:59.989014N
South	53:09:00.005493S
East	177:07:48.676758E
West	165:16:12.015381W
3D	no

History
COMMAND: v.in.ogr -o dsn="/home/gsherman/desktop_gis_data/cities.shp" output="cities" min_area=0.0001 snap=-1
GISDBASE: /home/gsherman/gsherman-dapper/grassdata/
LOCATION: world_geo MAPSET: gsherman
USER: gsherman DATE: Sun Aug 26

As we work with GRASS maps, a history is recorded. You can see in our browser example the command used to create the cities layer from the original shapefile. This is displayed in the right pane, just below the extent information. Using the browser gives you a quick overview of your maps and layers, as well as some detailed information about the number of features and the history of the map.

The toolbar in the browser has a number of useful functions for working with and managing our maps (layers). You can add a layer to the map canvas, copy a map, rename a map, delete a map, set the current region to that of the selected map, and refresh the browser tree.

The browser is a handy tool when working with and managing your GRASS data.

8.6 Analysis

Most people are aware that GRASS has some powerful features. The QGIS-GRASS plugin makes access to those features easier and also facilitates learning about what's going on behind the scenes. This in turn can help you make the transition to using the modules from the GRASS command line.

If you look through the modules in the GRASS toolbox you'll see that there are a large number of tools available for working with your GRASS data. In this section, we'll look at a few examples of what you can do with the GRASS geoprocessing tools.

8.6.1 Creating a Buffer

A common GIS operation is to create a buffer. You might want to examine the impact of development or establish "setbacks" to protect a critical resource. With the GRASS toolbox we can quickly create a buffer using `v.buffer`.

Exercise 22 Creating a Buffer

Let's create a buffer of the `cities` layer:

1. Open your mapset containing the layer
2. Load the `cities` layer
3. Open the GRASS toolbox
4. Under Buffer, click on the `v.buffer` module
5. Enter a buffer distance of 0.1
6. Give the output map a name: `cities_buffer`
7. Click Run
8. Add the buffer layer to the map and examine it

Specifying a buffer distance in decimal degrees may not be intuitive. For many geoprocessing operations you'll want to work in a projected coordinate system, for a number of reasons.

8.6.2 Creating Contours

Another module allows us to create a contour map from a raster containing elevation or other data. We can use the raster we imported earlier into our GRASS mapset to create contours:

Exercise 23 Creating Contours from the DEM

1. Open your mapset containing the layer
2. Load the `ancc6_geo` raster layer
3. Open the GRASS toolbox
4. Find `r.contour` under the "Raster" heading and click on it
5. Make sure `ancc6_geo` is selected in the drop-down box
6. Click on the Use region of this map button to the right of the drop-down box
7. Enter the increment between contours—we'll use 100
8. Enter a name for the output raster: `ancc6_contour`
9. Click Run
10. Add the contour layer to the map and examine it

This created a contour map with 100 meter contour intervals. We know this because the original DEM has elevations in meters.

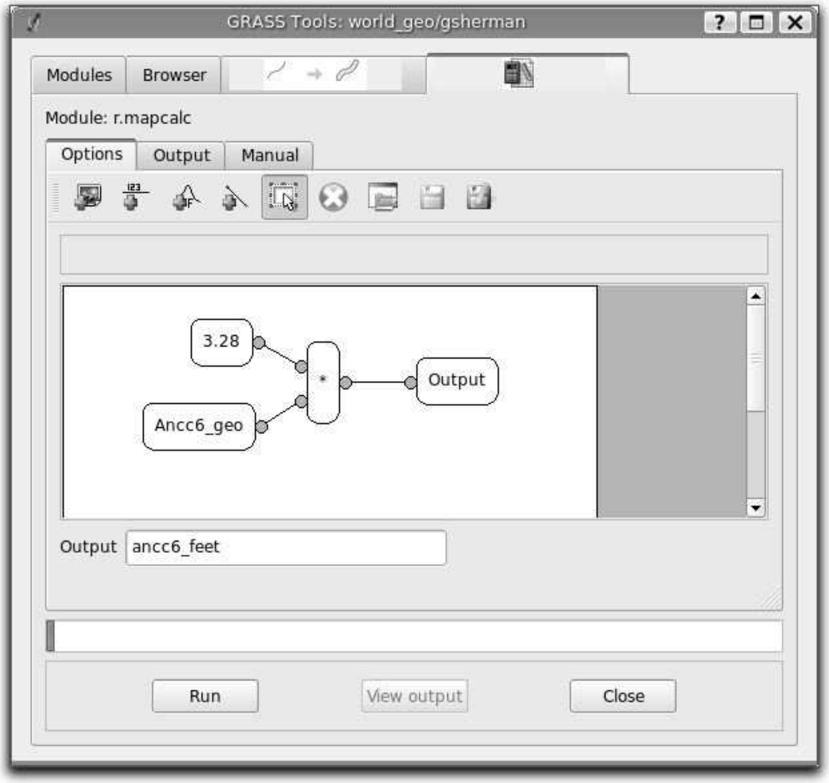
8.6.3 Map Algebra

The toolbox contains the "Raster map calculator". This allows you to create a model and run it to perform operations on each cell in a raster. You could also use it to combine two rasters to create a third, using simple addition or a more complicated operation or function.

Exercise 24 Using Map Algebra

Let's use the calculator to convert our DEM from meters to feet:

1. Open your mapset containing the layer
2. Load the `ancc6_geo` raster layer
3. Open the GRASS toolbox
4. Click on the "Browser" tab
5. Find the `ancc6_geo` layer and click on it
6. Click on the (Set current region to selected map) tool. This constrains the raster processing to the region of our original DEM
7. Click on the "Modules" tab
8. Find the `r.mapcalc` module
9. Create the model shown in the graphic below using the tools in the calculator. Make sure you select `ancc6_geo` before adding the map to the model. In addition to the map, you need to add a constant (3.28) and a multiplication operator to the model.
10. Once the model is complete, enter a name for the output raster
11. Click [Run] to run the model
12. Add the new raster to the map and examine it



Module: r.mapcalc

Output ancc6_feet

We used a very simple operation—multiplication—to create a new raster with cell values in feet. You may have noticed a lot of functions available in the tool. This gives you a lot of power to construct complicated models for raster processing.

8.7 Summary

Our look at QGIS-GRASS integration just touched on the many options available to you. Using QGIS to work with your GRASS data is a good way to get started using GRASS. Many people have found that they soon are using GRASS outside of QGIS and being quite successful at performing complex operations and geoprocessing not yet supported by the toolbox.

Tip 9 Learning More About GRASS

To learn more about GRASS and the commands available through the QGIS GRASS toolbox, be sure to look at the "Manual" tab for each module as you use it. Also be sure to visit the GRASS website at `http://grass.itc.it` for more documentation and tutorials.

Chapter 9

Python Plugins

Writing plugins in Python is much simpler than using C++. To create a PyQGIS plugin, you need QGIS 0.9, Python, PyQt, and the Qt developer tools.

When QGIS starts up it scans certain directories looking for both C++ and Python plugins. For a file (shared library, DLL, or python script) to be recognized as a plugin it has to have a specific signature. For Python scripts it's pretty simple. QGIS looks in the following locations under the installation directory:

1. Linux and other Unix: ./share/qgis/python/plugins
2. Mac OS X: ./Contents/MacOS/share/qgis/python/plugins
3. Windows: .\share\QGIS\python\plugins

Each Python plugin is contained in its own directory. When QGIS starts up it will scan each subdirectory in `share/qgis/python/plugins` and initialize any plugins it finds. Once that's done, the plugin will show up in the plugin manager.

Let's create a plugin to fill a gap in the QGIS interface. This plugin will allow us to create a new PostGIS layer for us to digitize. It will be a simple plugin and pretty rough, but it illustrates how to get started writing your own PyQGIS plugins.

9.1 Setting up the Structure

The first thing we need to do is set up the structure for our plugin. In this example we'll be developing our plugin on Linux but the method is the same, just adapt some of the file system commands as appropriate for your platform. QGIS is installed in a directory named `qgis_09` in our home directory. Let's create the directory for the plugin.

```
mkdir ~/qgis_09/share/qgis/python/plugins/new_layer
```

To get started, we need to create the following files in the `new_layer` directory (we'll need some additional files in a bit):

1. __init__.py
2. resources.py
3. resources.qrc
4. newlayer.py

9.2 Making the Plugin Recognizable

Initializing the plugin is done in the `__init__.py` script. For our `NewLayer` plugin the script contains:

```
# load NewLayer class from file newlayer.py
from newlayer import NewLayer
def name():
  return ”New PostGIS layer”
```

```
5 def description ():
6   return ”Creates a new empty Postgis layer”
7 def version ():
8   return ”Version 0.1”
9 def classFactory( iface ):
10  return NewLayer(iface)
```

The mandatory things a plugin must return are a name, description, and version, all of which are implemented in our script above. Each method simply returns a string with the appropriate information. The other requirement is the *classFactory* method that must return a reference to the plugin itself (line 10), after receiving the **iface** object as an argument. With this simple code, QGIS will recognize our script as a plugin.

9.3 Resources

In order to have a nice icon for our plugin, we need a resources file which we'll name `resources.qrc`. This is just a simple XML file that defines the icon resource:

```
 <RCC>
    <qresource prefix=”/plugins/newlayer”>
        <file>icon.png</file>
    </qresource>
</RCC>
```

The resource file uses a prefix to prevent naming clashes with other plugins—using the name of the plugin is usually sufficient. The `icon.png` file is is just a PNG image that will be used in the toolbar when the plugin is activated. You can use any image, as long as it's 22x22 pixels (so it fits on the toolbar).

To turn the resource file into something the plugin can use, it must be compiled using the PyQt resource compiler:

```
pyrcc4 -o resources.py resources.qrc
```

The `-o` switch is used to specify the output file. Now that we have resources, we need a way to collect the information needed for creating a new layer.

9.4 Creating the GUI

Normally we would use the same tool that C++ developers use to create a GUI: Qt Designer. This is a visual design tool that allows you to create dialog and main windows by dragging and dropping widgets and defining their properties.

To design our NewLayer plugin we could get quite fancy and include widgets for field types and other options. However, since our time is limited, we'll use another means to collect the information we need to create the table. This will illustrate the concepts and then you can venture further using the tutorials on the QGIS blog.

To collect the user input, we'll use the **QInputDialog** class from the Qt library. This prompts the user for a single line of input. While it will make our plugin a little crude, it serves to illustrate the concepts.

All we need to write now is the Python code to collect the input and create the table.

9.5 Creating the Plugin

Now that we have the preliminaries out of the way, we can get down to writing the code that does the actual work. Let's start by looking at the things we need to import and the initialization of the plugin in `newlayer.py`.

```
1 # Import the PyQt and QGIS libraries
2 from PyQt4.QtCore import ∗
3 from PyQt4.QtGui import ∗
4 from qgis.core import ∗
5 import psycopg
6 # Initialize Qt resources from file resources.py
7 import resources
8
9 # Our main class for the plugin
10 class NewLayer:
11
12   def __init__ ( self , iface ):
13     # Save reference to the QGIS interface
14     self . iface = iface
15
16   def initGui( self ):
17     # Create action that will start plugin configuration
18     self .action = QAction(QIcon(”:/plugins/newlayer/icon.png”),\
19       ”New PosGIS Layer”, self.iface.getMainWindow())
20     QObject.connect(self.action, SIGNAL(”activated()”), self.run)
21
22     # Add toolbar button and menu item
23     self . iface .addToolBarIcon(self.action)
24     self . iface .addPluginMenu(”&New PostGIS Layer...”, self.action)
25
26   def unload(self ):
27     # Remove the plugin menu item and icon
28     self . iface .removePluginMenu(”&New PostGIS Layer...”,self.action)
29     self . iface .removeToolBarIcon(self.action)
```

In lines 2 through 7 we import the libraries needed for the plugin. This includes the PyQt libraries, the QGIS core library, and the Python PostgreSQL library psycopg. Every Python script that uses the QGIS libraries and PyQt needs to import the QtCore and QtGui libraries, as well as the QGIS core library. This gives us access to the PyQt wrappers for our Qt objects (like our input dialog) and the QGIS core libraries. We also need to import the `resources.py` file we created with the icon definition.

In line 10 we declare the class **NewLayer**. In the *__init__* method (lines 12 through 14) our class is initialized and passed the **iface** object from QGIS via the *classFactory* method in line 10 of __init__.py. We store **iface** as a member variable so we can use it later.

In lines 16 through 24 we initialize the GUI elements for the plugin. In Qt, a **QAction** is used to create a user interface action that can be used to create both a menu and toolbar item. In our plugin, we use it for both. In line 18 we create the action using our icon resource (note the prefix we specified in `resources.qrc`). We also provide some text that will appear when it is used in a menu or during a mouseover, and lastly we need to specify the “parent”. In a plugin, the parent is the main window of QGIS. The **iface** object that we stored during initialization allows us to get the reference to the main window in line 19.

Once the action is created, we can add it to both the toolbar and the Plugins menu (lines 23 and 24). That takes care of initializing the GUI for the plugin. The other thing we need to do is clean up after ourself when the plugin is unloaded. The *unload* method takes care of this by removing the menu item and the tool from the toolbar (lines 28 and 29).

This takes care of the initialization stuff and getting our plugin to load and unload nicely. Now let’s look at the code that does the actual work. It’s all contained in the *run* method.

```
30  def run(self):
31    # Get the user input, starting with the table name
32    table_name = QInputDialog.getText(None, ”Table Name?”, \
33      ”Name for new PostGIS layer”)
34    if table_name[0].length() > 0:
35      # Get the field names and types
36      fields = QInputDialog.getText(None, ”Field Names”, \
37        ”Fields (separate with a comma)”)
38      parts = fields [0]. split ( ’,’ )
39      # Create the SQL statement
40      sql = ”create table ” + table_name[0] + ” (id int4 primary key, ”
41      for fld in parts:
42        sql += fld + ” varchar(10), ”
43      sql = sql[0:−2]
44      sql += ”)”
45      # Connect to the database
46      # First get the DSN
47      dsn = QInputDialog.getText(None, ”Database DSN”, \
48        ”Enter the DSN for connecting to the database (dbname=db user=user)”)
49      if dsn[0].length() > 0:
50        con = psycopg.connect(str(dsn[0]))
51        curs = con.cursor()
52        curs.execute(str(sql))
53        con.commit()
54        # add the geometry column
55        curs.execute(”select AddGeometryColumn(’” + str(table_name[0]) + \
56          ”’, ’the_geom’, 4326, ’POLYGON’, 2)”)
57        con.commit()
58        # create the GIST index
59        curs.execute(”create index sidx_” + str(table_name[0]) + ” on ” + \
60          str(table_name[0]) + ” USING GIST(the_geom GIST_GEOMETRY_OPS)”)
61          con.commit()
```

The first thing we need to do is use the **QInputDialog** to get the name of the table to create. This is done in line 32 where we prompt for it.

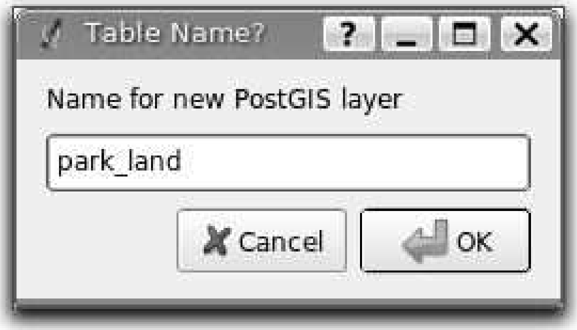


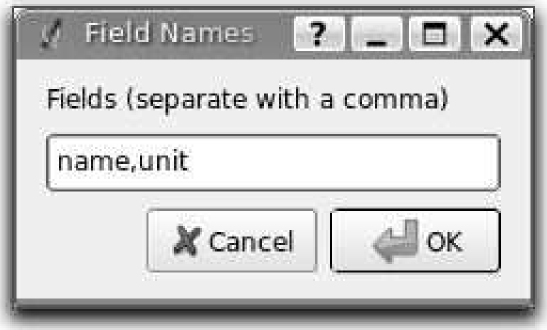
Table Name?

Name for new PostGIS layer

park_land

In line 34 we check to see if the user actually entered anything before proceeding.

Next we need to get the field names. For this example we are keeping it very simple. Every field will be a varchar(10), meaning we can store up to 10 characters in it. If we really want to make this plugin useful, we would need to provide a way for the user to specify the type. In line 36 we prompt the user to enter a comma delimited list of field names.



Field Names

Fields (separate with a comma)

name,unit

We then split this list into its components for use in constructing the SQL statement (line 38).

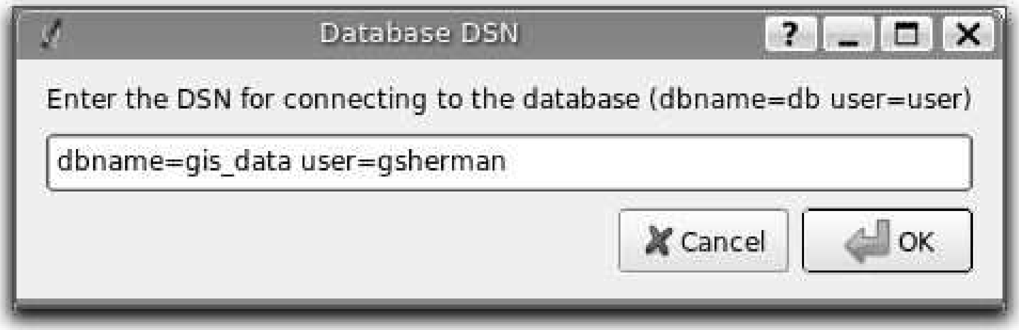
Line 40 contains the first part of the SQL statement. Note we are creating the table with an integer id field that will be the primary key. We then iterate through the field list, appending the appropriate code to the SQL statement (line 41).

Once we have all the fields added to the SQL statement, we chop off the trailing characters we don't want (line 43) and then add the closing parenthesis to complete the statement (line 44).

Now we are ready to connect to the database and create the table. To access the database, we are using psycopg (`http://www.initd.org`). In order to connect we need to specify the data source name (DSN) with the name of the database, the user, and a password if necessary. If we are running both QGIS and PostgreSQL on the same machine we usually don't need to specify a password. In this case, the DSN will look something like this:

```
dbname=gis_data user=gsherman
```

To get the DSN, we prompt the user with a **QInputDialog** in line 47.


Database DSN

Enter the DSN for connecting to the database (dbname=db user=user)

dbname=gis_data user=gsherman

If the user enters a DSN then we can proceed with the connection to the database in line 50. We get a cursor from the connection in line 51 and then execute the SQL statement to create the table and commit the change in lines 52 through 53. This creates the table, but for it to be a valid layer and ready for us to use it needs a couple more things.

First it needs a geometry column. We purposely didn't include one when we created the table so we could use the *AddGeometryColumn* function to create it. This function adds a geometry column to the table and then puts an entry in the **geometry_columns** table for us. In line 55 we specify the table name, the name we want for the geometry column, the SRID, feature type, and the dimension of the feature.

The last thing to do is create a spatial index on the table so we get optimum performance when doing spatial searches and displaying the data in QGIS. In line 59 we have cobbled together the SQL to create the index. The actual statement looks like this:

```
create index sidx_park_land on park_land
   USING GIST(the_geom GIST_GEOMETRY_OPS);
```

9.6 Issues and Problems

Our plugin is now complete. Now lets look at some of the things that are wrong with it or where we could improve it:

- We could use an improved GUI, one that lets the user enter all the needed information on one dialog
- The user can't specify field types
- There is limited error checking in the dialog
  - If you don't enter any fields, the plugin fails
  - There is no error checking on any of the database operations
- There is no feedback from the plugin once it completes

With all the issues, it still servers as a primordial plugin that illustrates the process and helps get you started with your own plugin development.

9.7 Adding Feedback

Let's fix one of the small problems by adding some feedback at the end of the process. We'll just add a message box to tell the user that everything is done and to check the database to make sure the table was created.

To do this, we just add the following code after line 61:

```
62          # show the user what happened
63          QMessageBox.information(None, "Results", "Table " + str(table_name[0]) + \
64            " has been created. Check your database to confirm.")
```

When the table is created, the user sees this:

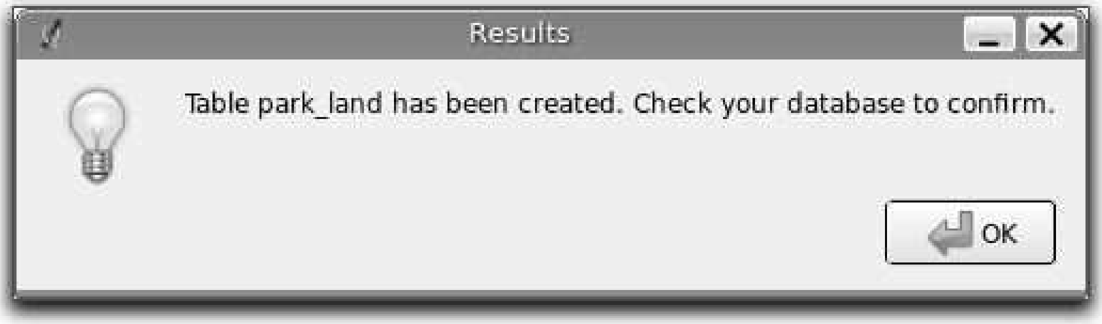


Table park_land has been created. Check your database to confirm.

9.8 Summary

Writing a QGIS plugin in Python is pretty easy. Some plugins won't require a GUI at all. For example, you might write a plugin that returns the map coordinates for the point you click on the map. Such a plugin wouldn't require any user input and could use a standard Qt **QMessageBox** to display the result.

You can also write plugins for QGIS in C++, but that's another story. You can find tutorials on writing QGIS plugins in both C++ and Python on the QGIS blog at:

`http://blog.qgis.org`

Chapter 10

Creating 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. With the release of 0.9, development of standalone applications using either C++ or Python is possible.

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 PyQGIS[1] 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.

10.1 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
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`

[1] An application created using Python and the QGIS bindings

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.

10.2 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 # Import our resources (icons)
16 import resources
17
18 # Environment variable QGISHOME must be set to the 0.9 install directory
19 # before running this application
20 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 resources file on line 16.

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 0.9. 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)
```

```

        # Set the title for the app
        self .setWindowTitle(”FOSS4G2007 Demo App”)

        # Create the map canvas
        self .canvas = QgsMapCanvas()
        # Set the background color to light blue something
        self .canvas.setCanvasColor(QColor(200,200,255))
        self .canvas.enableAntiAliasing(True)
        self .canvas.useQImageToRender(False)
        self .canvas.show()

        # Lay our widgets out in the main window using a
        # vertical box layout
        self .layout = QVBoxLayout(self.frame)
        self .layout.addWidget(self.canvas)

        # Create the actions for our tools and connect each to the appropriate
        # method
        self .actionAddLayer = QAction(QIcon(”:/foss4g2007/mActionAddLayer.png”),
        \
            ”Add Layer”, self.frame)
        self .connect(self .actionAddLayer, SIGNAL(”activated()”), self.addLayer)
        self .actionZoomIn = QAction(QIcon(”:/foss4g2007/mActionZoomIn.png”), \
            ”Zoom In”, self.frame)
        self .connect(self .actionZoomIn, SIGNAL(”activated()”), self.zoomIn)
        self .actionZoomOut = QAction(QIcon(”:/foss4g2007/mActionZoomOut.png”), \
            ”Zoom Out”, self.frame)
        self .connect(self .actionZoomOut, SIGNAL(”activated()”), self.zoomOut)
        self .actionPan = QAction(QIcon(”:/foss4g2007/mActionPan.png”), \
            ”Pan”, self .frame)
        self .connect(self .actionPan, SIGNAL(”activated()”), self.pan)
        self .actionZoomFull = QAction(QIcon(”:/foss4g2007/mActionZoomFullExtent.png”), \
            ”Zoom Full Extent”, self.frame)
        self .connect(self .actionZoomFull, SIGNAL(”activated()”),
        self .zoomFull)

        # Create a toolbar
        self .toolbar = self .addToolBar(”Map”)
        # Add the actions to the toolbar
        self .toolbar.addAction(self.actionAddLayer)
        self .toolbar.addAction(self.actionZoomIn)
        self .toolbar.addAction(self.actionZoomOut);
        self .toolbar.addAction(self.actionPan);
        self .toolbar.addAction(self.actionZoomFull);

        # Create the map tools
        self .toolPan = QgsMapToolPan(self.canvas)
        self .toolZoomIn = QgsMapToolZoom(self.canvas, False) # false = in
        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 our resources file. 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.setFillColor(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.

10.3 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.

```
def main(argv):
  # create Qt application
  app = QApplication(argv)

  # Initialize qgis libraries
  QgsApplication.setPrefixPath(qgis_prefix, True)
  QgsApplication.initQgis()

  # create main window
  wnd = MainWindow()
  # Move the app window to upper left
  wnd.move(100,100)
  wnd.show()

  # run!
  retval = app.exec_()

  # exit
  QgsApplication.exitQgis()
  sys.exit(retval)


if __name__ == ”__main__”:
  main(sys.argv)
```

10.4 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. On Linux or OS X:

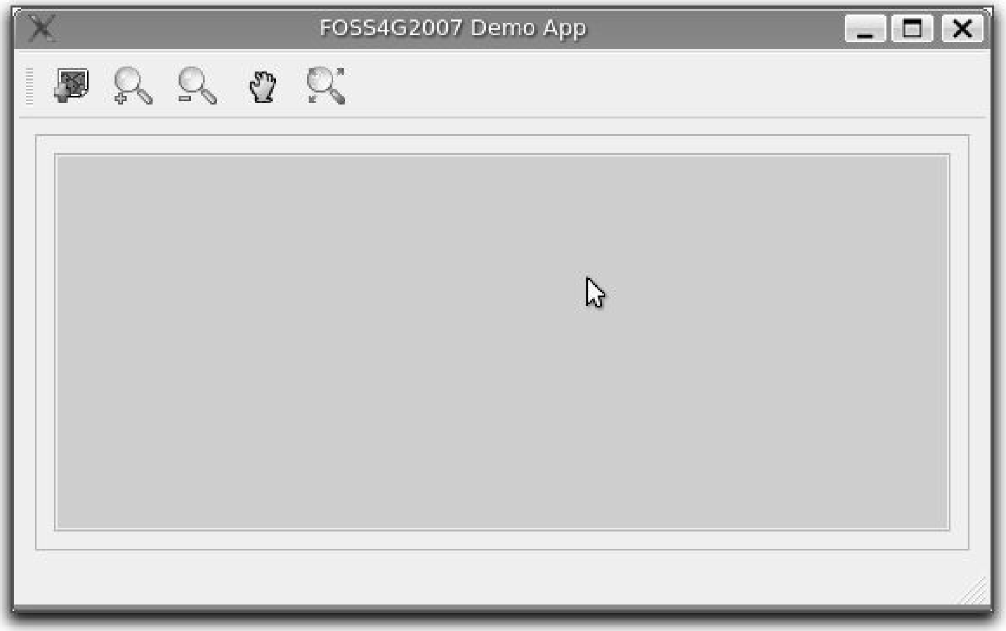
```
export LD_LIBRARY_PATH=$HOME/qgis_09/lib
export PYTHONPATH=$HOME/qgis_09/share/qgis/python
export QGISHOME=$HOME/qgis_09
```

For Windows:

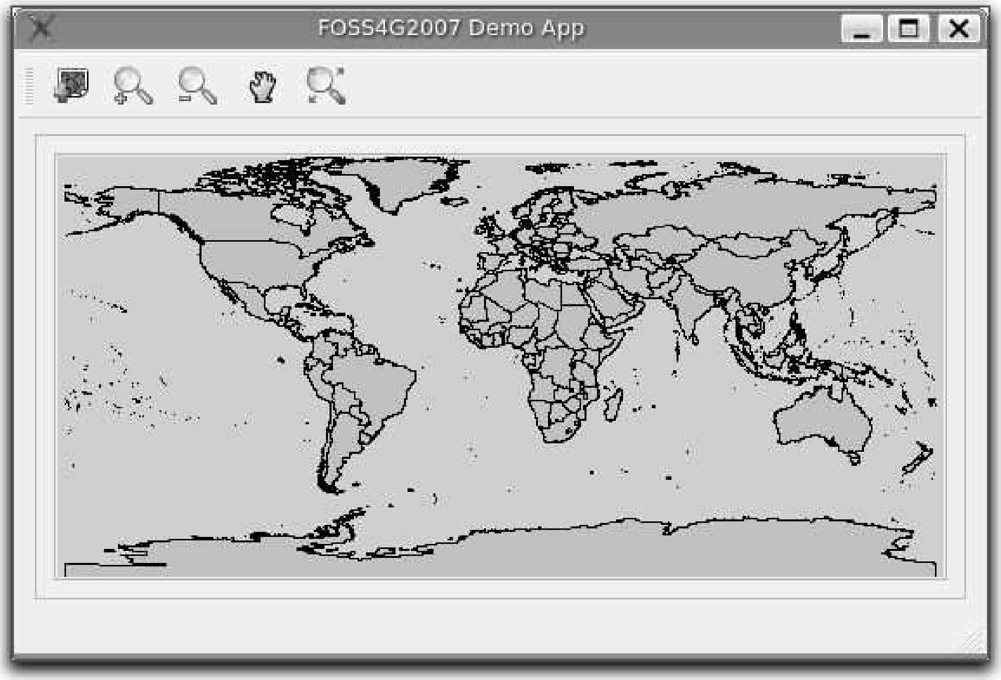
```
set PATH=C:\qgis;%PATH%
set PYTHONPATH=C:\qgis\python
set QGISHOME=C:\qgis
```

In the case of Linux or OS X, we assume that QGIS is installed in your home directory in `qgis_09`. For Windows, 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 Open 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 Space 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 0.9.

Tip 10 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://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.

Appendix A

Installing QGIS

To install QGIS, point your web browser at `http://qgis.org` and find the downloads section. There you will find packages for the major platforms, including Linux, Mac OS X, and Windows.

You have several options for installing, depending on your platform:

1. Install using the Mac OS X framework packages from kyngchaos.com
2. Use the Windows installer
3. Install the deb or rpm for your Linux version
4. Build from source

QGIS depends on a number of other open source packages. When installing on Mac OS X and Windows, these are bundled for you in the framework packages or the installer. On Linux and other *nix platforms, you will need to install the dependencies prior to installing or building QGIS. For distributions that have a package manager, this is usually not a difficult task. Its best to look around before building dependencies from scratch to see if packages or builds for your operating system are available.

A complete list of dependencies and build instructions for the current version can be found at `http://qgis.org`.