Quantum GIS Design Document

QGIS Core Design Team*

26th August 2005

*Gary E Sherman, Mark Coletti, Denis Antipov

Contents

1 Introduction

This document describes the requirements and design for Quantum GIS (QGIS), a desktop GIS application for Linux and Unix. This document presents use cases, high-level class diagrams, and additional information about the design and implementation of QGIS.

QGIS is hosted on SourceForge at http://qgis.sourceforge.net. The current release of QGIS is a viewer with a minimal feature set, including loading data, browsing, and identifying features.

The design outlined in this document represents the next phase of QGIS development, which will move the application to a flexible and extensible platform for working with spatial data.

Note that it's presumed that the reader is familiar with C++, object-oriented design, and UML.

2 Definitions

The following is a list of definitions for terms used in this document. Some of these definitions are taken from the Association for Geographic Information and the University Of Edinburgh Department of Geography online dictionary of GIS terms available at http://www.geo.ed.ac.uk/agidict/welcome.html.

Attribute: A fact describing an entity in a relational data model, equivalent to the column in a relational table.

Feature: A set of points, lines or polygons in a spatial database that represent a real-world entity. The terms feature and object are often used synonymously.

GIS: Geographic Information System. A software system that provides the ability to view and analyze spatial data and its attributes.

Layer: A dataset that has a spatial context and can be drawn on a map canvas. Layers may have associated attributes.

Plugin: A compiled library that can be dynamically loaded into an application to provide new functionality.

Projection: A method of representing the earth's three-dimensional surface as a flat two-dimensional surface. This normally involves a mathematical model that transforms the locations of features on the earth's surface to locations on a two-dimensional surface. Because the earth is three-dimensional, some method must be used to depict the map in two dimensions. Therefore such representations distort some parameter of the earth's surface, be it distance, area, shape, or direction.

3 History

The QGIS project was registered with SourceForge on June 15, 2002. Since that time, QGIS has developed into a minimally functional viewer with support for shapefiles[1] and vector data stored in a PostgreSQL[2] database using the PostGIS[3] extensions.

The development thus far has been useful in developing an understanding of the challenges involved in developing a more robust Open Source desktop GIS application. In March 2003, planning began to restructure QGIS in order to make it more extensible and to provide a means to add advanced functionality through the use of plugins.

The current version of QGIS (0.0.9) is still usable as a simple GIS viewer for shapefiles and PostGIS layers. Only minor maintenance is being done on the current code base at this time.

[1] ESRI format for file-based spatial data.
[2] PostgreSQL Relational Database - http://www.postgresql.org
[3] PostGIS extension - http://postgis.refractions.net

4 Goals

There are already other Open Source GIS projects available today. Many have asked what purpose QGIS will serve:

- Will QGIS be a complete desktop GIS application?
- Will it compete with commercial products?
- Why are you developing another GIS application?

The answers to these questions are not clear-cut. A list of high-level goals for the QGIS project are enumerated below. The reader can perhaps use this information to answer these and other questions related to the project:

4.1 List of Goals

1. Provide an easy to use desktop GIS application
2. Provide an easy to install application for users with minimum system experience
3. Support common data formats
4. Provide the foundation for more advanced tools (plugins)
5. Become a tool for spatial data collection
6. Support data analysis and conversion
7. Print/plot maps
8. Integrate with Internet mapping technologies

Section 5 provides details about the functional requirements.

5 Requirements

This section describes the functional requirements of Quantum GIS. These functional requirements drive the use cases discussed in Section 6.

QGIS will provide GIS functionality somewhere between a simple viewer and an industrial strength application. The ultimate nature of QGIS is only limited by the talent of those software engineers who will provide advanced capabilities through plugins.

Some of the major design requirements include:

- Extensible architecture using plugins
- Internationalization
- Integrated scripting language
- Projection on-the-fly
- Flexible symbology for all feature classes
- Ability to render and browse data in many formats:
  - Spatio-temporal data using a feature-centric model
  - Shapefiles
  - PostgreSQL / PostGIS layers

  - Raster

- Support for OpenGIS implementation Specifications

  - Geography Markup Language (GML)
  - Web Feature Service (WFS)

5.1 User Interface

QGIS will use the SDI (Single Document Interface). QGIS currently has (and will have) standard GUI interface elements such as menus, toolbar, and a statusbar. In addition, the standard QGIS interface will have a legend panel and a map canvas (or drawing area).

5.2 Standards

Development of QGIS will proceed with adoption of applicable OpenGIS standards. This will include support for GML and possibly WFS.

5.3 Data Formats

Any data format could be supported by the development of a plugin that reads and renders the data store. The "standard" formats that will be included in the core QGIS are similar to those currently supported:

- Shapefiles
- PostgreSQL/PostGIS
- Rasters

Format support will be provided by plugins.

5.4 Platform Support

Initially, QGIS will target Linux and other Unix operating systems supported by the Qt toolkit. Coding during development of QGIS and plugins will be done in a way so as to not introduce any platform dependencies. This will provide to possibility of a Windows version at some point in the future.

5.5 Toolkits

QGIS will leverage existing libraries and toolkits to support the GUI, GIS data stores, and GIS processing algorithims. At present the identified toolkits include:

- Qt (http://www.trolltech.com)
- GDAL and OGR (/http://www.remotesensing.org/gdal)
- Proj.4 (http://www.remotesensing.org/proj/)

6 Use Cases

Use cases provide a means to identify and visualize the major goal oriented tasks the application should address.

6.1 Actors

Actors are really persons (or physical entities) that use a system to acheive a specific goal (these goals are expressed as use cases). A number of actors could be defined for QGIS, however at this point the simple approach is taken. The actors are:

- Casual GIS User
- Professional GIS User

These two actors are sufficient to frame the development and discussion of QGIS use cases.

6.2 Primary Use Cases

The primary use cases for QGIS are listed below in no particular order of importance:

- Load data
- Browse data
- Install plugin
- Find feature
- dislayFeatureAttributes
- Get help
- Customize
- Save project
- Restore project
- Navigate map
  - Pan
  - Zoom
- Save image
- Print
  - Print image
  - Print metadata
  - Print feature information
- Write script
- Run script
- Edit data
  - Digitize
- Buffer feature
- Process data
  - Union
  - Merge
  - Intersect
- Convert data
  - Change projection
- Import data
- Export data
  - Export selected set
- Select features
  - Select by attribute
  - Select spatially
- Edit Layer Preferences
  - Edit Symbology
  - Edit Labels

6.3 Case Descriptions

In the sections that follow, the use cases are presented in no particular order.

6.3.1 Load Data

Use Case:	Load Data
Goal in Context:	Load a data set from a data set
Scope & Level:	Primary task
Preconditions:	Application is running
Success End Condition:	Data is loaded and displayed
Failed End Condition:	Application is in pre-load state
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to load data into environment for use

Description of Steps:

User	System
selects data	
	acknowledges
start load	
	system shows loading of data then shows data

6.3.2 Browse Data

Use Case:	Browse Data
Goal in Context:	Browse through a loaded data set and its features
Scope & Level:	Primary task
Preconditions:	Application is running, data set loaded
Success End Condition:	Ability to browse, look at the needed information about loaded data
Failed End Condition:	Unable to browse and look at the loaded data information
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to browse the features and other information about the data set

Description of Steps:

User	System
Change View	
	Redisplay View
<<extension point>>	
[continue until done]	

6.3.3 Load Plugin

Use Case:	Load plugin
Goal in Context:	Load a plugin into the 'core' system form the available list of installed plugins
Scope & Level:	Primary task
Preconditions:	Application is running
Success End Condition:	Plugin loaded and the the new tools corresponding to the plugin appear available
Failed End Condition:	Application is in the initial or the previous plugin state
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to view data with a different plugin, or wants to add certain functionality to the existing state

Description of Steps:

User	System
Ask for list of available plugins	
	Shows plug-ins
Selects one	
	System loads that plug-in

6.3.4 Find Feature

Use Case:	Find feature
Goal in Context:	Find a feature on a map or loaded data set
Scope & Level:	Primary task
Preconditions:	Application is running and a data set loaded
Success End Condition:	A feature that corresponds to the search condition found if such exists and not found if not exists
Failed End Condition:	A feature that exists and corresponds to the search conditions not found
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to find a feature on a map or data set

Description of Steps:

User	System
Select feature	
	Show feature

6.3.5 displayFeatureAttributes

Use Case: displayFeatureAttributes

Goal in Context: After selecting a feature on a map, display its attributes

Scope & Level: Primary task

Preconditions: Application is running and a data set loaded

Success End Condition: Attributes of a feature displayed

Failed End Condition: Nothing displayed when a feature selected

Primary Actor(s): Casual, Professional user

Trigger: User wants to view attributes of a feature

Description of Steps: User decides to view attributes of a particular feature
User chooses a feature on the map
User selects a feature
Attributes are displayed

User	System
<select feature>	
	feature shown to be selected
ask for attributes	
	display attributes attached to feature

6.3.6 Get Help

Use Case:	Get Help
Goal in Context:	Load help system
Scope & Level:	Primary task
Preconditions:	Application is running
Success End Condition:	Help system loaded and the menu of help topics displayed
Failed End Condition:	Application is in 'helpless' state
Primary Actor(s):	Casual, Professional user
Trigger:	User needs help with an application or one of its plugins
Description of Steps:	1. User finds out that they don't know as much about QGIS as they thought they did 2. User launches help system 3. User selects one of the topics/subtopics of interest 4. User is enlightened about rich QGIS functionality and how to use it

6.3.7 Customize

Use Case:	Customize
Goal in Context:	Customize application user interface
Scope & Level:	Primary task
Preconditions:	Application is running a plugin that is being customized loaded
Success End Condition:	Interface changes according to customization
Failed End Condition:	No changes occus after customization
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to change the interface
Description of Steps:	1. User decides to change the interface 2. User launches the customization system 3. User adjusts the interface according to the customizationsystem 4. Changes in the interface take effect

6.3.8 Save Project

Use Case:	Save Project
Goal in Context:	Save entire project in a specified file format
Scope & Level:	Primary task
Preconditions:	Application is running, data set(s) loaded
Success End Condition:	Current project saved in a file
Failed End Condition:	Current project not saved in a file
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to save the layout of map, layers, etc.
Description of Steps:	1. User decides to save the current project 2. User chooses location and filename for the project 3. The project saved in a file

6.3.9 Restore Project

Use Case:	Restore Project
Goal in Context:	Load a project from a previously saved file
Scope & Level:	Primary task
Preconditions:	Application is running
Success End Condition:	Project loaded and all the data sets restored in the saved condition
Failed End Condition:	Project is not restored
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to restore the project that was saved
Description of Steps:	1. User decides to restore a project 2. User navigates to the location of the project 3. User selects a project 4. Project is restored

6.3.10 Navigate

Use Case:	Navigate
Goal in Context:	Pan and Zoom on a data set
Scope & Level:	Primary task
Preconditions:	Application is running, data set loaded and rendered
Success End Condition:	Data set is zoomed or panned
Failed End Condition:	No changes to the view of a data set
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to zoom or pan on a data set
Description of Steps:	1. User decides to zoom or pan on a data set 2. User uses zoom or pan 3. The view is zoomed or panned

6.3.11 Save Image

Use Case:	Save Image
Goal in Context:	Save the current view of the data set in an image file
Scope & Level:	Primary task
Preconditions:	Application is running, data set loaded and rendered
Success End Condition:	An image displaying the current view of a data set saved
Failed End Condition:	No image saved
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to save the current view of a data set as an image
Description of Steps:	1. User decides to save an image 2. User chooses the location and filename of the image 3. The image saved

6.3.12 Print

Use Case:	Print
Goal in Context:	Output on a printer the current data set as an image, metadata, and its set of features
Scope & Level:	Primary task
Preconditions:	Application is running, a dataset loaded and rendered
Success End Condition:	The different types of data (image, metadata, feature set) being output on a printer
Failed End Condition:	No attempt to output data on a printer
Primary Actor(s):	Casual, Professional user
Trigger:	User wants to output data on a printer
Description of Steps:	1. User decides to output data set on a printer 2. User chooses the type of data output (image, metadata, feature set) 3. User selects a printer 4. Dataset is output on a printer in a specified format

6.3.13 Write Script

Use Case:	Write Script
Goal in Context:	Write a script to perform some QGIS function in an automated manner
Scope & Level:	Primary task
Preconditions:	Application is running
Success End Condition:	Script is completed and tested
Failed End Condition:	Script is not saved and application is in pre-script state
Primary Actor(s):	Professional user
Trigger:	User needs a script to perform a repititve or specialized operation
Description of Steps:	1. Identify need for the script 2. Open the script editor 3. Write and test script in iterative fashion 4. Save script for future use

6.3.14 Run Script

Use Case:	Run Script
Goal in Context:	Execute a stored script to perform a task
Scope & Level:	Primary task
Preconditions:	Application is running with appropriate or required data loaded
Success End Condition:	Script performs desired function
Failed End Condition:	Script exits gracefully
Primary Actor(s):	Casual, Professional user
Trigger:	User needs to perform a repetitive or complex task for which a script exists
Description of Steps:	1. Identify script that can be used to perform the desired task 2. Select the script 3. Additional requirements necessary to execute the script are met 4. Execute script 5. Result of script is reported to the user or displayed on the map canvas

6.3.15 Edit Data

Use Case:	Edit Data
Goal in Context:	Edit data to change the attributes, spatial location, or shape of a feature or features
Scope & Level:	Primary task
Preconditions:	Data layer to be edited has been loaded
Success End Condition:	Data is changed and saved to the data store
Failed End Condition:	Data is not changed but remains in the state prior to beginning the edit operation
Primary Actor(s):	Professional user
Trigger:	User needs to modify data
Description of Steps:	1. Enter edit mode 2. Locate feature to be modified 3. Edit the feature 4. Save data 5. Exit edit mode

6.3.16 Digitize

Use Case:	Digitize Data
Goal in Context:	Create a new data layer or add to an existing layer by digitizing information from a raster image on the screen (heads-up digitizing)
Scope & Level:	Primary task
Preconditions:	Raster is loaded and displayed
Success End Condition:	New features are added to the data layer and saved to the data store
Failed End Condition:	Existing data is not changed but remains in the state prior to beginning the edit operation
Primary Actor(s):	Professional user
Trigger:	User needs to create new data from a raster
Description of Steps:	1. Loads or create the target layer 2. Enter edit mode 3. Digitize features using the underlying raster 4. Add attributes to each feature as it is completed 5. Save data 6. Exit edit mode

6.3.17 Buffer Feature

Use Case:	Buffer Feature
Goal in Context:	Create a new data layer that contains a buffer around one or more features on an existing layer
Scope & Level:	Primary task
Preconditions:	Data layer containing features to buffer is loaded and displayed
Success End Condition:	New data layer containing polygon buffers is created
Failed End Condition:	No new data layer is created and application remains in state prior to beginning of buffer operation
Primary Actor(s):	Professional user
Trigger:	User needs to buffer features for the purpose of performing spatial analysis or answering a question about proximity of features
Description of Steps:	1. Select feature or features to buffer 2. Specify the buffer distance in appropriate map units 3. Create buffer 4. Save buffer layer to permanent store

6.3.18 Process Data - Union

Use Case:	Process Data - Union
Goal in Context:	Create a new data layer that contains the union of two existing layers
Scope & Level:	Primary task
Preconditions:	Data layers needed for operation are loaded and displayed
Success End Condition:	New data layer containing union of source layers is created
Failed End Condition:	No new data layer is created and application remains in state prior to beginning of union operation
Primary Actor(s):	Professional user
Trigger:	User needs to union features for the purpose of performing spatial analysis
Description of Steps:	1. Select layers to union 2. Perform union operation 3. Save resulting layer to permanent store

6.3.19 Process Data - Intersect

Use Case:	Process Data - Intersect
Goal in Context:	Create a new data layer that contains the intersection of two existing layers
Scope & Level:	Primary task
Preconditions:	Data layers needed for operation are loaded and displayed
Success End Condition:	New data layer containing intersection of source layers is created
Failed End Condition:	No new data layer is created and application remains in state prior to beginning of intersect operation
Primary Actor(s):	Professional user
Trigger:	User needs to create a new layer representing the intersection of features from two source layers
Description of Steps:	1. Select layers to intersect 2. Perform intersect operation 3. Save resulting layer to permanent store

6.3.20 Process Data - Merge

Use Case:	Process Data - Merge
Goal in Context:	Create a new data layer that contains the contents of two existing layers. The layers are "tiled" into one.
Scope & Level:	Primary task
Preconditions:	Data layers needed for operation are loaded and displayed
Success End Condition:	New data layer containing merge of source layers is created
Failed End Condition:	No new data layer is created and application remains in state prior to beginning of merge operation
Primary Actor(s):	Professional user
Trigger:	User needs to combine the features from two source layers into one
Description of Steps:	1. Select layers to merge 2. Perform merge operation 3. Save resulting layer to permanent store

6.3.21 Change projection

Use Case:	Change Projection
Goal in Context:	Project a source data layer into a new map projection to create a new data layer
Scope & Level:	Primary task
Preconditions:	Data layer needed for operation is loaded and displayed
Success End Condition:	Data layer in desired projection is created from the source layer
Failed End Condition:	No new data layer is created and application remains in state prior to beginning of merge operation
Primary Actor(s):	Professional user
Trigger:	User needs to a copy of a data layer in a different projection
Description of Steps:	1. Select layer to project 2. Specify the projection of the source layer 3. Specify the projection for the new layer 4. Project the data 5. Save resulting layer to permanent store

6.3.22 Import Data

Use Case:	Import Data
Goal in Context:	Create a new data layer by importing data from a source not directly usable with QGIS
Scope & Level:	Primary task
Preconditions:	Application is running
Success End Condition:	New data layer is created
Failed End Condition:	No new data layer is created and application remains in state prior to beginning of import operation
Primary Actor(s):	Professional user
Trigger:	User needs to work with data not currently in a format supported by QGIS
Description of Steps:	1. Select layer to project 2. Specify the projection of the source layer 3. Specify the projection for the new layer 4. Project the data 5. Save resulting layer to permanent store

6.3.23 Export Data

Use Case:	Export Data
Goal in Context:	Convert data from one format (source) to another (targert) for the purpose of using it in another application.
Scope & Level:	Primary task
Preconditions:	Data layer needed for operation is loaded and displayed
Success End Condition:	Export file is created
Failed End Condition:	Export file is not created and application remains in state prior to beginning of export operation
Primary Actor(s):	Professional user
Trigger:	User needs to export data for use by another application
Description of Steps:	1. Select data layer to export 2. Choose output location 3. Export the data

6.3.24 Export Selected Set

Use Case:	Export Selected Set
Goal in Context:	Convert a selected set of feature data from one format (source) to another (targert) for the purpose of using it in another application.
Scope & Level:	Primary task
Preconditions:	Data layer needed for operation is loaded and displayed
Success End Condition:	Export file is created
Failed End Condition:	Export file is not created and application remains in state prior to beginning of export operation
Primary Actor(s):	Professional user
Trigger:	User needs to export data for use by another application
Description of Steps:	1. Select data layer to export 2. Select features to be included in the export 3. Choose output location 4. Export the features

6.3.25 Select Spatially

Use Case:	Select Spatially
Goal in Context:	Select features on the map canvas for purpose of using in another function or operation
Scope & Level:	Subfunction
Preconditions:	Data layer needed for operation is loaded and displayed
Success End Condition:	Features are selected and highlighted on map
Failed End Condition:	No selection set is created and application remains in state prior to beginning of export operation
Primary Actor(s):	Professional user
Trigger:	User needs to select features for use in another function or operation
Description of Steps:	1. Select data layer containing features of interest 2. Interactively draw a box around features 3. Select

6.3.26 Select by Attribute

Use Case:	Select by Attribute
Goal in Context:	Select features on the map canvas for purpose of using in another function or operation
Scope & Level:	Subfunction
Preconditions:	Data layer needed for operation is loaded and displayed
Success End Condition:	Features are selected and highlighted on map
Failed End Condition:	No selection set is created and application remains in state prior to beginning of export operation
Primary Actor(s):	Professional user
Trigger:	User needs to select features for use in another function or operation
Description of Steps:	1. Select data layer containing features of interest 2. Open the query builder to define the selection criteria 3. Execute the query to select features

6.3.27 Edit Layer Preferences

Use Case:	Edit Layer Preferences
Goal in Context:	Edit the name of the layer and other attributes that determine how it is rendered.
Scope & Level:	Primary Task
Preconditions:	Data layer user desires to edit preferences for is loaded and displayed
Success End Condition:	Changes are reflected in the display of the layer
Failed End Condition:	Layer properties remain unchaged or are not changed to the desired values
Primary Actor(s):	Casual, Professional user
Trigger:	User wishes to give layer a name and styling to aid in better organizing and viewing the project
Description of Steps:	1. Select data layer 2. Open the preferences dialog for that layer 3. Edit settings as desired 4. Apply changes and close dialog

6.3.28 Edit Symbology

Use Case:	Edit Symbology
Goal in Context:	Choose what symbols will be used to render features in a layer for the layer as a whole, or based on attributes of the features.
Scope & Level:	Subfunction
Preconditions:	Layer Preferences dialog is open
Success End Condition:	Changes are reflected in the display of the layer
Failed End Condition:	Layer is not rendered using the chosen symbology
Primary Actor(s):	Professional user
Trigger:	User wishes to use color, fill type, line styles and other symbology to reveal more detailed information in a shapefile layer
Description of Steps:	1. Edit settings as desired 2. Apply changes and close dialog

6.3.29 Edit Labels

Use Case:	Edit Labels
Goal in Context:	Show feature labels based on attribute tables or choose alternate labels to display
Scope & Level:	Subfunction
Preconditions:	Layer preferences dialog is open
Success End Condition:	The layer is rendered with the chosen labels
Failed End Condition:	Layer is rendered with no labels or without using the specified labels
Primary Actor(s):	Professional user
Trigger:	User wishes to control what labels appear for layer features
Description of Steps:	1. Edit settings as desired 2. Apply changes and close dialog

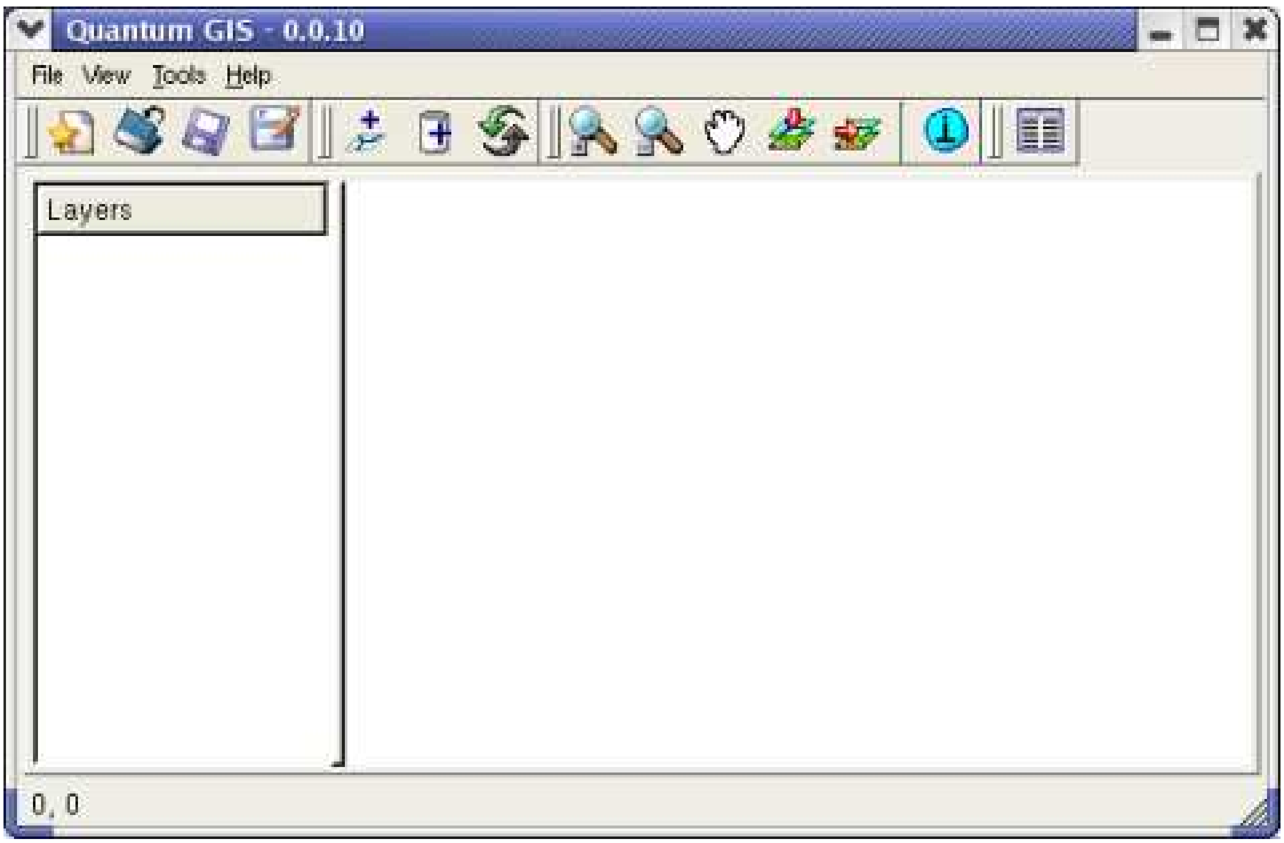
7 Core Architecture

This section describes the functionality of the QGIS "core". The core application contains minimal functionality. QGIS depends on plugins to implement all non-trivial functions. The components of the core are discussed in the following sections. Details with regard to physical implementation are included where appropriate.

7.1 Main Window

The QGIS main window consists of a Qt QMainWindow widget and includes docking areas, menus, toolbar(s), and status bar area. The window is further divided into a legend panel and canvas panel. The general layout of the main window is shown below is shown in Figure 1. The size of the main windows, as well as the location of each toolbar is saved on exit and restored on startup.

Figure 1: Sample QGIS Main Window Layout



Plugins are free to implement additional GUI elements or windows to accomplish their task(s).

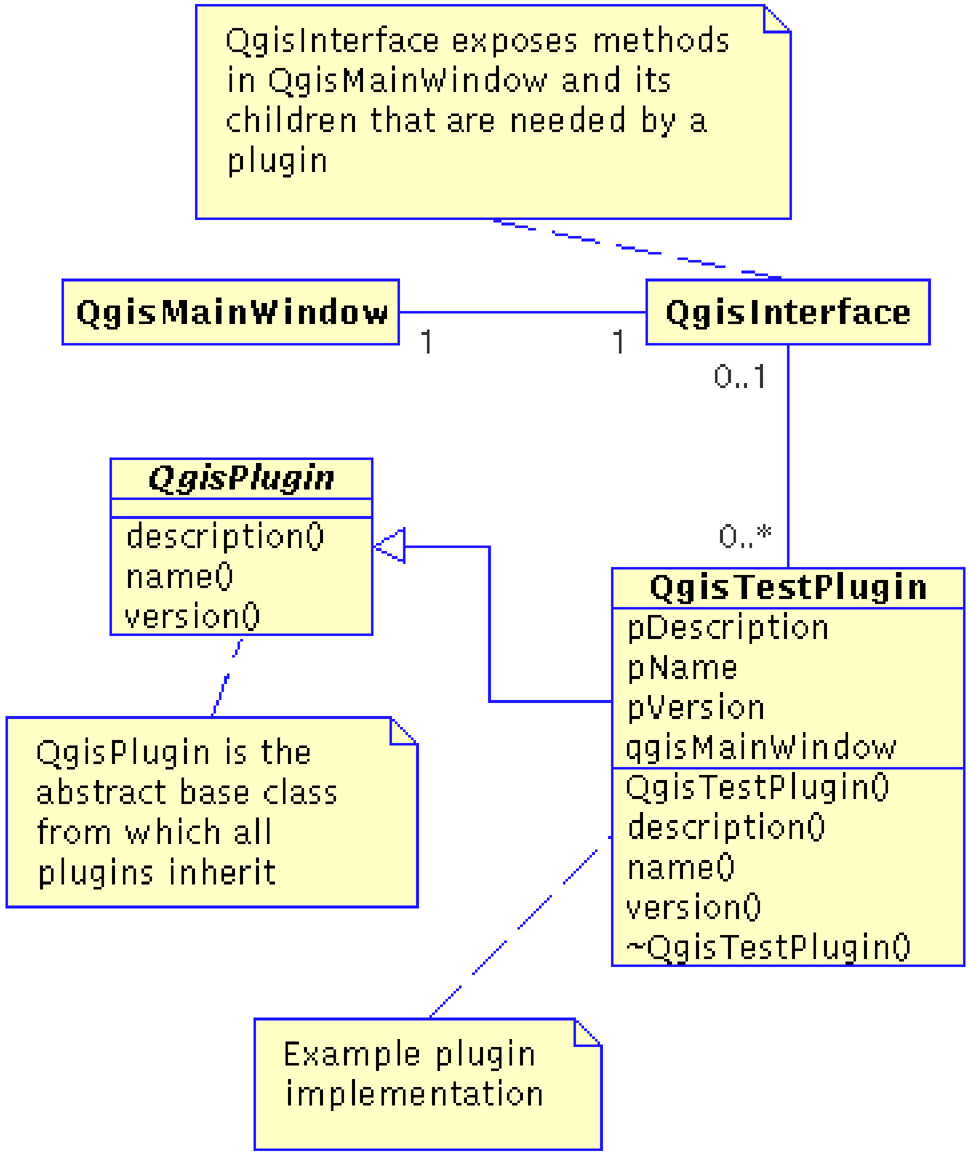
7.2 Plugins

QGIS will rely heavily on plugins to implement all major functions. With a flexible plugin architecture, QGIS can be easily extended to support additional features and capabilities.

7.2.1 Class Diagram

The class diagram in Figure 2 provides a conceptual model of QGIS with regard to plugins.

Figure 2: Conceptual Diagram of Plugin Architecture



7.2.2 Plugin Load Sequence

The main application class uses QLibrary to load and resolve plugin functions. In order for a plugin to be recognized, it must implement the methods defined in the QgisPlugin abstract class and also return a valid version string when queried by the main application. The sequence of events when a plugin is loaded is:

1. QGIS attempts to load the plugin's shared library using QLibary::load().

2. If load succeeds, the classFactory method is resolved using QLibrary::resolve(). The classFactory method is declared as extern "C" and is responsible for creating an instance of the plugin and returning a pointer to it.

3. If the classFactory method is resolved successfully, QGIS calls the method, passing a pointer to the one and only instance of QgisInterface.

4. Once the plugin instance is created, the plugin can do any initialization required, including adding menus and toolbars to the using the QgisInterface pointer.

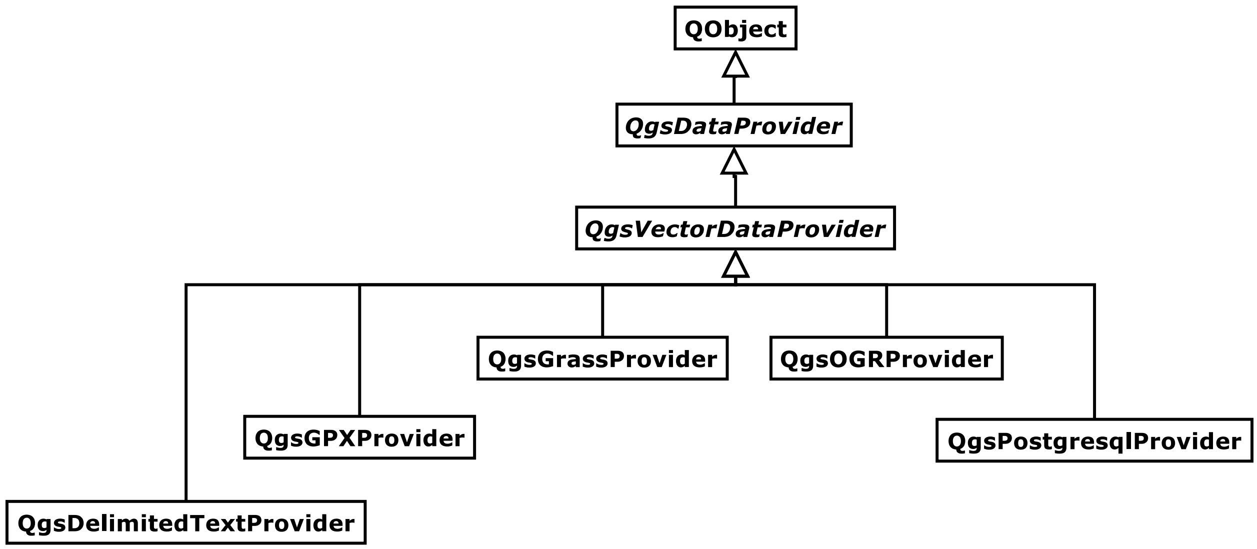


Figure 3: Data provider class hierarchy

7.3 Data Providers

Data providers are a special kind of plug-in. qgis uses data providers to implement geospatial data I/O for specific formats. The notion is that support for additional geospatial formats can easily added by merely creating a data provider plug-in that supports a given format. Moreover it should be possible to have multiple support for the same format; that is, have more than one data provider available to load a given geospatial data type. Figure 3 depicts the data provider class hierarchy.

Note that currently there is no `QgsRasterDataProvider` sub-hierarchy, though that will change in a subsequent design refactoring. Currently `QgsRasterLayer` manages its own data I/O directly through GDAL library calls.

The data provider plug-ins use a different loading method than for the general qgis plug-ins.

All data provider plug-in information is managed by the `QgsProviderRegistry` Singleton object. When initially created it searches the plug-in path for all data provider plug-ins. It does this by visiting each dynamic object in the path (`.DLL` in Windows, and `.so` in *nix environments); if the dynamic loader is able to resolve the symbol `isProvider`, then the currently loaded dynamic object is indeed a data provider. In that case, a `QgsProviderMetadata` object is created which stores a provider name, description, and full file name; this information is loaded from the dynamic object via calls to its `providerKey()`, `description()` and `library()` functions, respectively. The `QgsProviderRegistry` Singleton keeps an internal associative container of these `QgsProviderMetadata` objects keyed by the string returned from the providerKey() call. The alternative to having these `QgsProviderMetata` objects stored in this way would have been to load and keep *all* the data providers in memory during the application's lifetime regardless if they would ultimately be used. Instead of wastefully keeping all the data providers loaded, they are lazily loaded only when needed via the mechanism that uses `QgsProviderMetadata` objects. When a specific data provider is needed, the following steps are performed:

1. `QgsProviderRegistry::getProvider()` is invoked with a provider key
2. The `QgsProviderMetadata` matching that key is retrieved
3. A `QLibrary::load()` call is made with the associated file path found in the returned metadata `library()` call
4. The plug-in's `classFactory()` member is invoked with the data source name as its argument
5. The created data provider returned from the `classFactory()` invocation is then returned to the caller

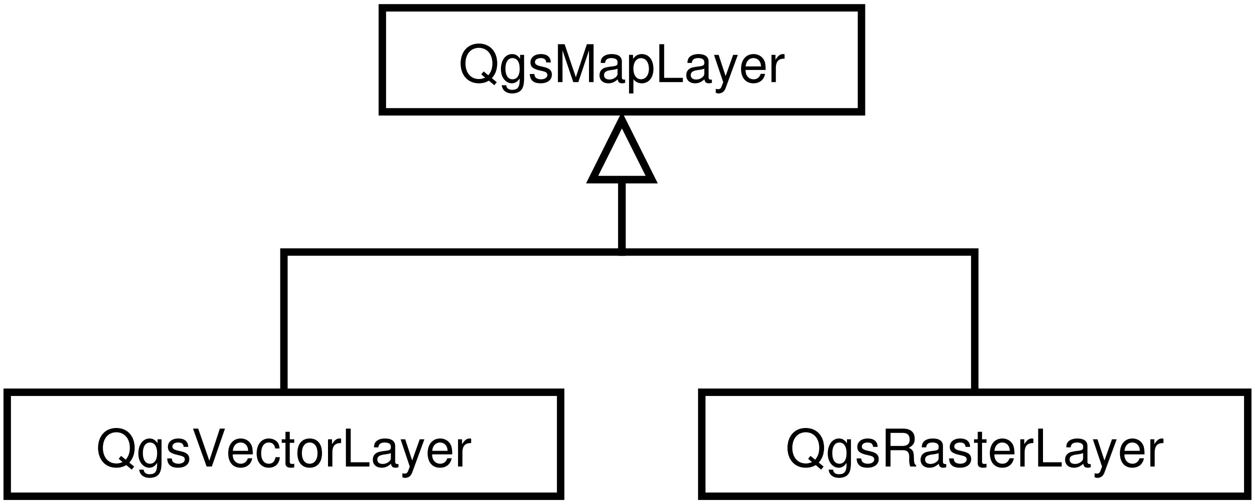


Figure 4: QgsMapLayer Class Hierarchy

The data provider is owned by its associated layer, which is then responsible for its destrction. (This will likely change in a subsequent refactoring to address the problem of a data source having more than one layer.)

7.4 Layers

As described on page 4 layer is a set of related geospatial data and their associated attributes. Layers tend to contain a particular type of data, such as hydrography, transportation, and hypsography data. Layers therefore tend to be "thematic" in nature. There are two kinds of qgis layers that follow from their representations – vector and raster layers. Figure 4 shows the `QgsMapLayer` hierarchy that qgis uses to implement vector and raster layers.

Layers have a unique identification string and a separate string for a base name. They also have a source string which can contain a file name or URI. The base name is used to identify the layer in the legend.

The `QgsMapLayerRegistry` is the Singleton object that stores all canonical layer instances. QgsMapLayerRegistry supports three Qt signals. `layerWasAdded()` is raised when a new layer is added to the registry. `layerWillBeRemoved()` is raised if a layer is to be deleted. `removeAll()` is raised if all the layers are to be cleared. The last is an optimization for clearing all layers at once instead of raising a signal for each layer that's removed when, say, clearing a project.

8 Use Case Scenario Design

In this section we describe the underlying design that supports the various use case scenarios given in section 6.

8.1 Load Data

This use case, as described on page 8, is of course for when the user loads data into qgis. Though vector and raster layers share many features, they are very different especially in how they respectively load data.

8.1.1 Vector Layer Loading

A new `QgsVectorLayer` is created when `QgisApp::addLayer()`, `QgisApp::addDatabaseLayer()`, or `QgsProject::read()` are invoked. The newly created `QgsVectorLayer` is given the file path for the data source, the basename of that file path as its name, and a data provider key used to find an appropriate data provider from the data provider registry described in section 7.3. The data provider key will be "ogr" if for a file based format since only the OGR data provider is the one currently supporting such. If PostGIS support is enabled, the data provider key can be "postgres" so the data provider registry can find the PostgreSQL/PostGIS data provider. (Yes, but this only happens when addDatabaseLayer is invoked. May need to further break this down or re-organize.)

8.1.2 Raster Layer Loading

Also triggered by QgisApp::addLayer() or QgsProject::read() a new `QgsRasterLayer` is created.