Quantum GIS 0.8
Plugin Writer Workshop

Dr. Marco Hugentobler
Institute for Cartography, ETH Zurich

Workshop Outline

- What are QGIS plugins?
- How QGIS plugins technically work
- Writing a simple plugin (point converter plugin)
  - step 1: Make QGIS recognize the plugin
  - step2: Add a button to the menu through the plugin
  - step3: Read the feature geometries of the current layer and store the points in a delimited text file
  - step4: Copy the feature attributes too

What are QGIS plugins?

- Dynamically linked libraries (.so or .dll)
- Are linked to QGIS at runtime when requested in the plugin manager
- Extend the functionality of QGIS
- Plugins have access to the QGIS GUI
- core plugins / external plugins
- Examples: GRASS, Del. text, GPS, SPIT

How QGIS plugins technically work

- The QGIS plugin manager looks in the lib/qgis directory for .so files
- The plugin manager loads all the .so files when it is open. When it is closed, the ones with a checked box are not unloaded
- To list the plugins, each plugin must have a few extern „C“ functions for description
- Once a plugin is selected in the list, the plugin manager calls the 'classFactory' method to create an instance of the plugin class

How QGIS plugins technically work

- The singleton class QgsPluginRegistry stores data about the loaded plugins
- The 'initGui' method of the plugin is called to show the GUI elements in the plugin menu and toolbar
- The unload() function of the plugin is to remove the allocated GUI elements
- The plugin itself is removed using the class destructor

PointConverter – a small sample plugin

- Searches the active layer in QGIS
- Converts all the vertices of the layer features to point features
- Keeps all the attributes
- Writes the point features into a delimited text file
- The new layer can then be loaded into QGIS using the delimited text plugin

Step 1: Make QGIS recognize the plugin

- Create QgsPointConverter.h/.cpp
- Add virtual methods inherited from QgisPlugin (but leave them empty for now)
- Create the needed extern „C“ methods
- Create a .pro file (Qt mechanism to easily create Makefiles)
- Compile
- Move the compiled library into the plugin folder
- Load the plugin in the QGIS plugin manager

Step 2: Add a button to the menu through the plugin

- Store a pointer to the QgisIface object in the plugin class
- Create a QAction and a callback function (slot)
- Add it to the QGIS GUI using QgisIface::addToolBarIcon() and QgisIface::addPluginMenu()
- Remove the QAction in the unload() method

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

- Query a location for the new file
- Iterate through all the features of the current layer
- Convert their geometrys to GEOS geometry
- Open a file and use a QTextStream
- Write x- and y-coordinates to the text file

Step 4: Copy the feature attributes too

- Extract the field vector using QgsVectorDataProvider::fields()
- For each feature, extract the field values using QgsFeature::attributeMap
- Add the contents of this map (comma separated) behind the coordinates for each new point features

Further information

- http://qgis.org for all QGIS related informations
- http://wiki.qgis.org/qgiswiki/DevelopingPlugins
- http://wiki.qgis.org/qgiswiki/DebuggingPlugins
- http://svn.qgis.org/api_doc/html/ (QGIS API doc)
- http://doc.trolltech.com/4.1/index.html (Qt doc)