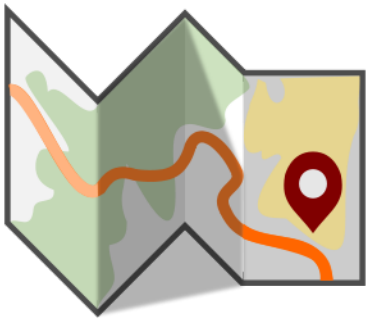
Building a Python Plugin

QGIS Tutorials and Tips



Author

Ujaval Gandhi

http://www.spatialthoughts.com

Building a Python Plugin

Plugins are a great way to extend the functionality of QGIS. You can write plugins using Python that can range from adding a simple button to sohpisticated toolkits. This tutorial will outline the process involved in setting up your development environment, designing the user interface for a plugin and writing code to interact with QGIS. Please review the *Getting Started With Python Programming* tutorial to get familiar with the basics.

Overview of the Task

We will develop a simple plugin called `Save Attributes` that will allow users to pick a vector layer and write its attributes to a CSV file.

Get the Tools

Qt Creator

Qt is a software development framework that is used to develop applications that run on Windows, Mac, Linux as well as various mobile operating systems. QGIS itself is written using the Qt framework. For plugin development, we will use an application called Qt Creator to design the interface for our plugin.

Download and install the Qt Creator software from SourgeForge

Python Bindings for Qt

Since we are developing the plugin in Python, we need to install the python bindings for Qt. The method for installing these will depend on the platform you are using. For building plugins we need the `pyrcc4` command-line tool.

Windows

Download the OSGeo4W network installer and choose Express Desktop Install. Install the package `QGIS`. After installation, you will be able to access the `pyrcc4` tool via the OSGeo4W Shell.

Mac

Install the Homebrew package manager. Install `PyQt` package by running the following command:

```
brew install pyqt
```

Linux

Depending on your distribution, find and install the `python-qt4` package. On Ubuntu and Debian-based distributions, you can run the following command:

```
sudo apt-get install python-qt4
```

A Text Editor or a Python IDE

Any kind of software development requires a good text editor. If you already have a favorite text editor or an IDE (Integrated Development Environment), you may use it for this tutorial. Otherwise, each platform offers a wide variety of free or paid options for text editors. Choose the one that fits your needs.

This tutorial uses Notepad++ editor on Windows.

Windows

Notepad++ is a good free editor for windows. Download and install the Notepad++ editor.

> ***Note***
>
> If you are using Notepad++, makes sure to check Replace by space at Settings ▸ Preferences ▸ Tab Settings. Python is very sensitive about whitespace and this setting will ensure tabs and spaces are treated properly.

Plugin Builder plugin

There is a helpful QGIS plugin named `Plugin Builder` which creates all the necessary files and the boilerplate code for a plugin. Find and install the `Plugin Builder` plugin. See *Using Plugins* for more details on how to install plugins.

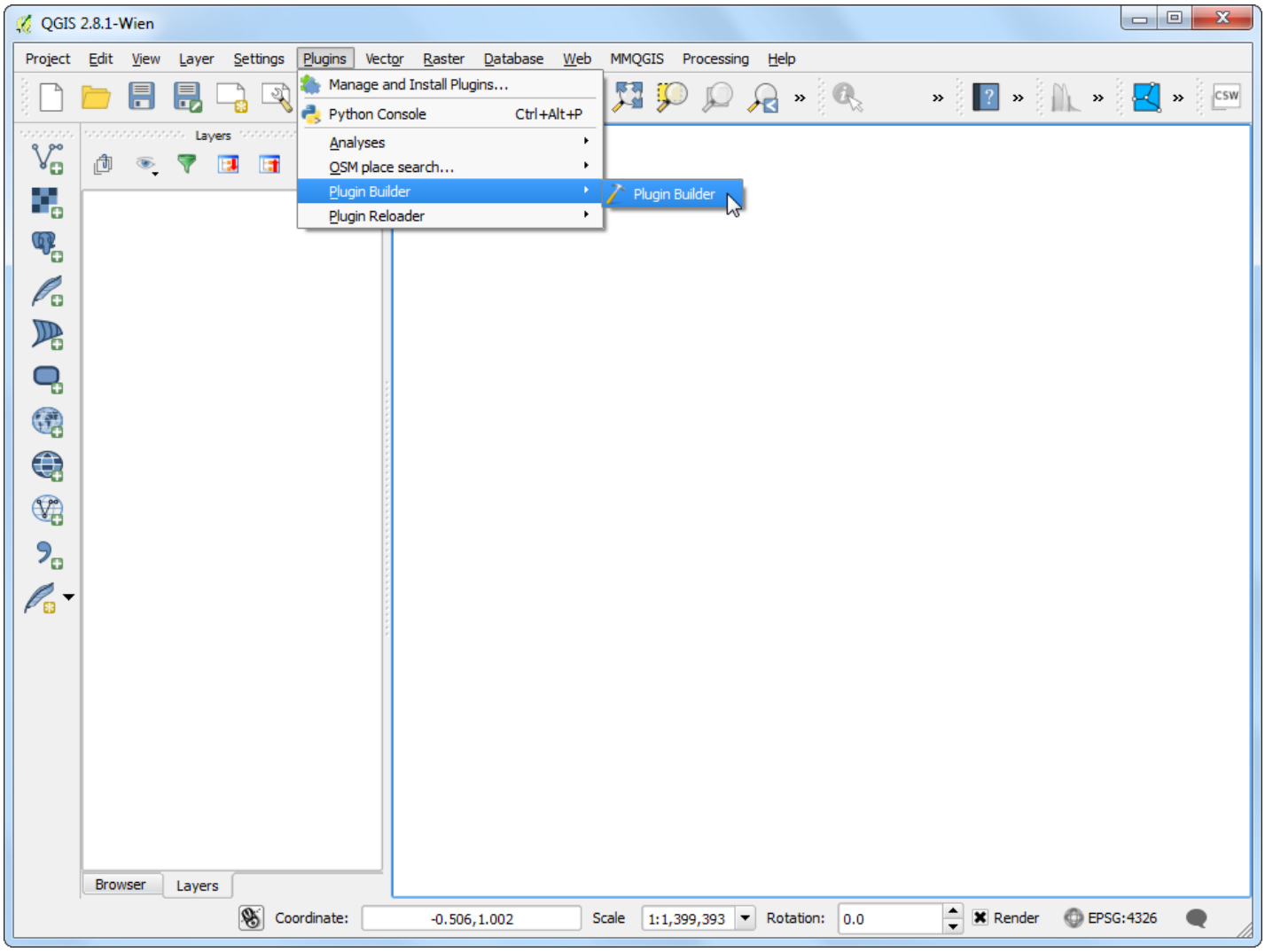
Plugins Reloader plugin

This is another helper plugin which allows iterative development of plugins. Using this plugin, you can change your plugin code and have it reflected in QGIS without having to restart QGIS every time. Find and install the `Plugin Reloader` plugin. See *Using Plugins* for more details on how to install plugins.

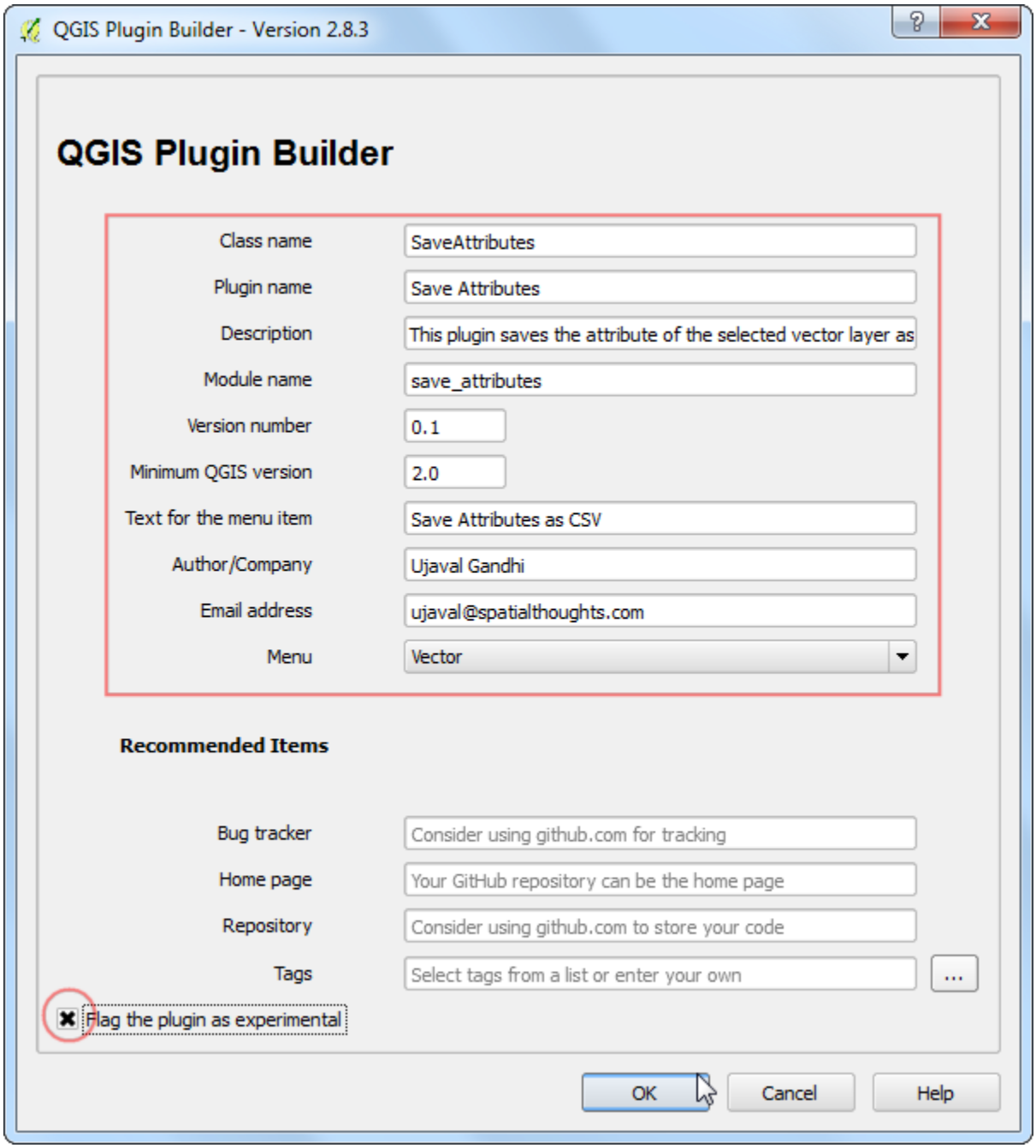
> ***Note***
>
> Plugin Reloader is an experimental plugin. Make sure you have checked Show also experimental plugins in Plugin Manager settings if you cannot find it.

Procedure

1. Open QGIS. Go to Plugins ▸ Plugin Builder ▸ Plugin Builder.



2. You will see the QGIS Plugin Builder dialog with a form. You can fill the form with details relating to our plugin. The Class name will be the name of the Python Class containing the logic of the plugin. This will also be the name of the folder containing all the plugin files. Enter `SaveAttributes` as the class name. The Plugin name is the name under which your plugin will appear in the Plugin Manager. Enter the name as `Save Attributes`. Add a description in the Description field. The Module name will be the name of the main python file for the plugin. Enter it as `save_attributes`. Leave the version numbers as they are. The Text for menu item value will be how the users will find your plugin in QGIS menu. Enter it as `Save Attributes as CSV`. Enter your name and email address in the appropriate fields. The Menu field will decide where your plugin item is added in QGIS. Since our plugin is for vector data, select `Vector`. Check the Flag the plugin as experimental box at the bottom. Click OK.



3. Next, you will be prompted to choose a directory for your plugin. You need to browse to the QGIS python plugin directory on your computer and select Select Folder. Typically, a `.qgis2/` directory is located in your home directory. The `plugin` folder location will depend on your platform as follows: (Replace `username` with your login name)

Windows

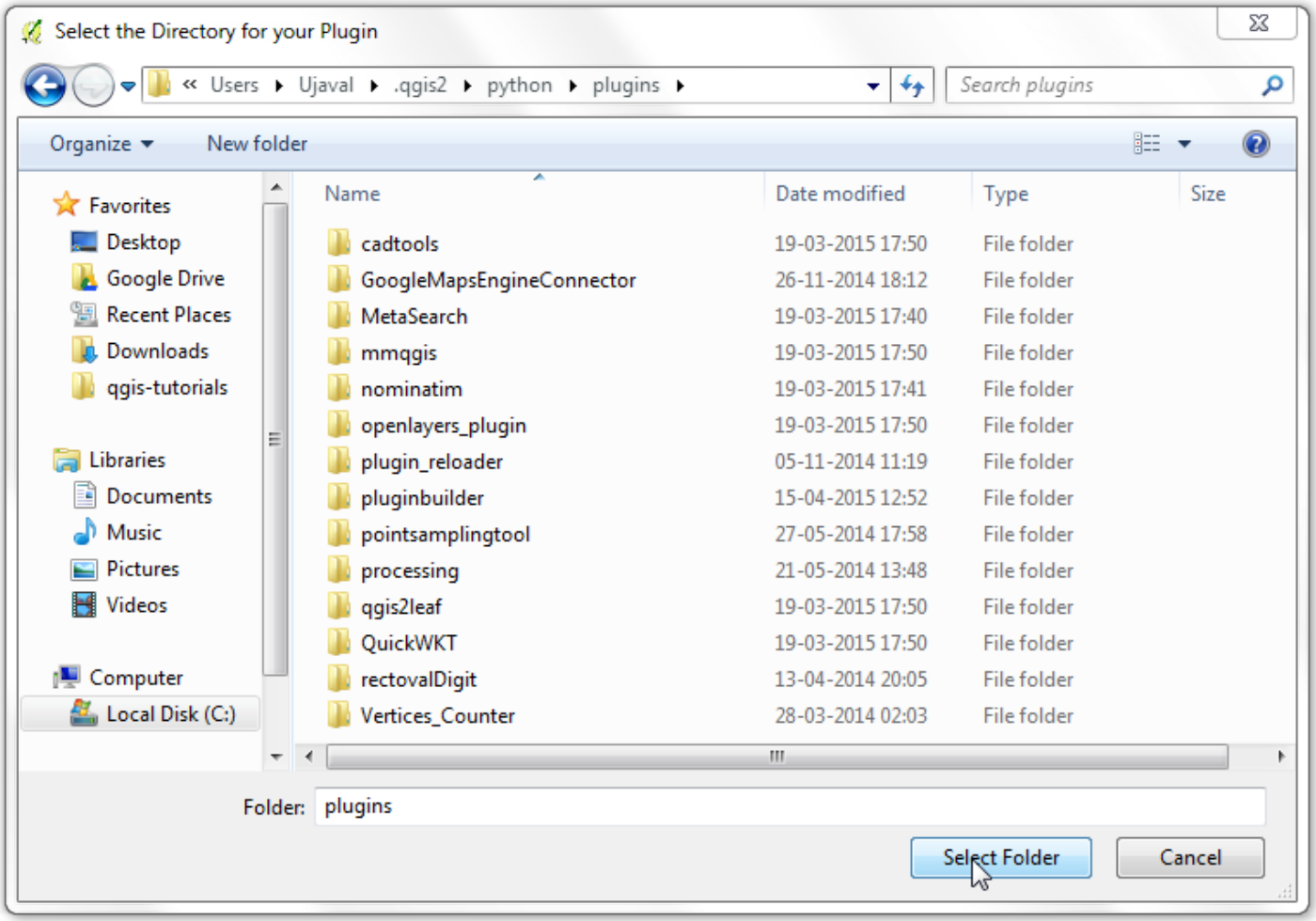
```
c:\Users\username\.qgis2\python\plugins
```

Mac

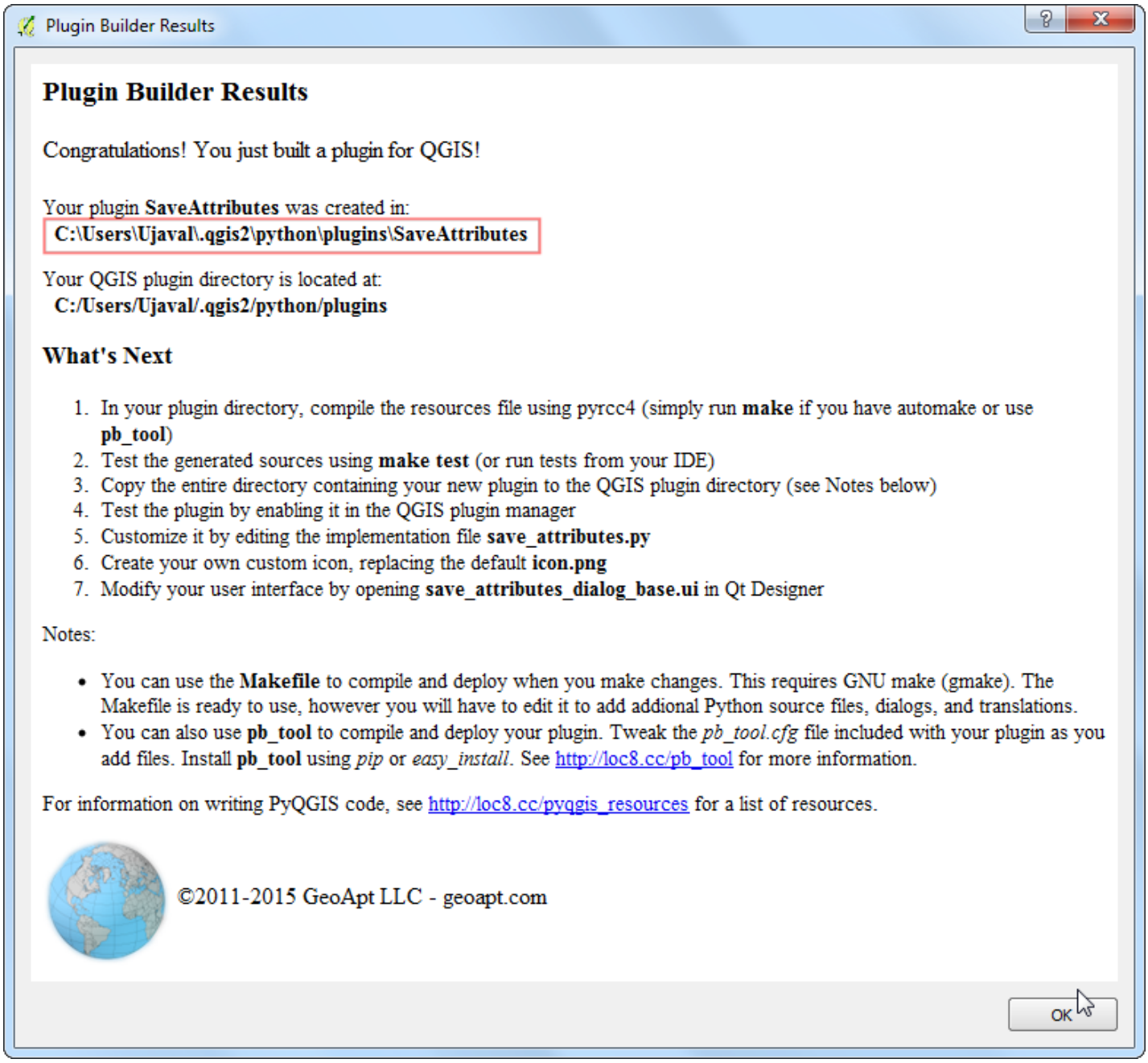
```
/Users/username/.qgis2/python/plugins
```

Linux

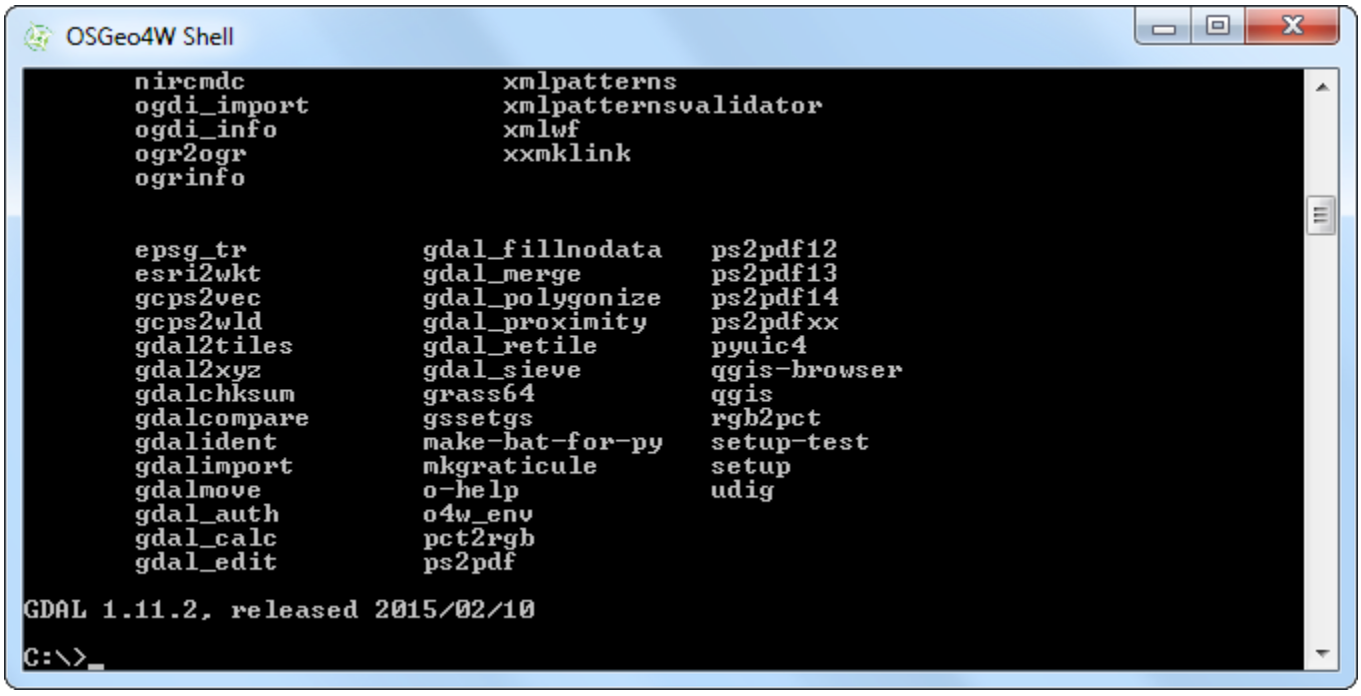
```
/home/username/.qgis2/python/plugins
```



4. You will see a confirmation dialog once your plugin template is created. Note the path to the plugin folder.

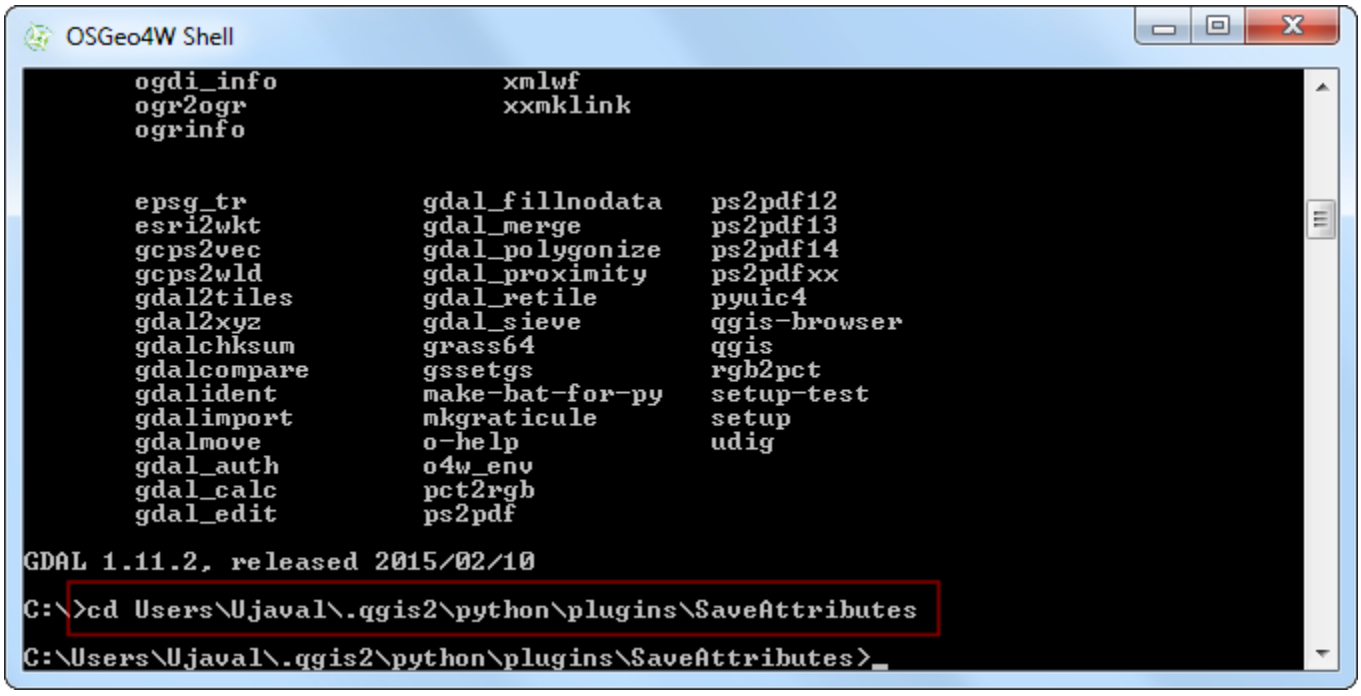


5. Before we can use the newly created plugin, we need to compile the `resources.qrc` file that was created by Plugin Builder. Launch the OSGeo4W Shell on windows or a terminal on Mac or Linux.



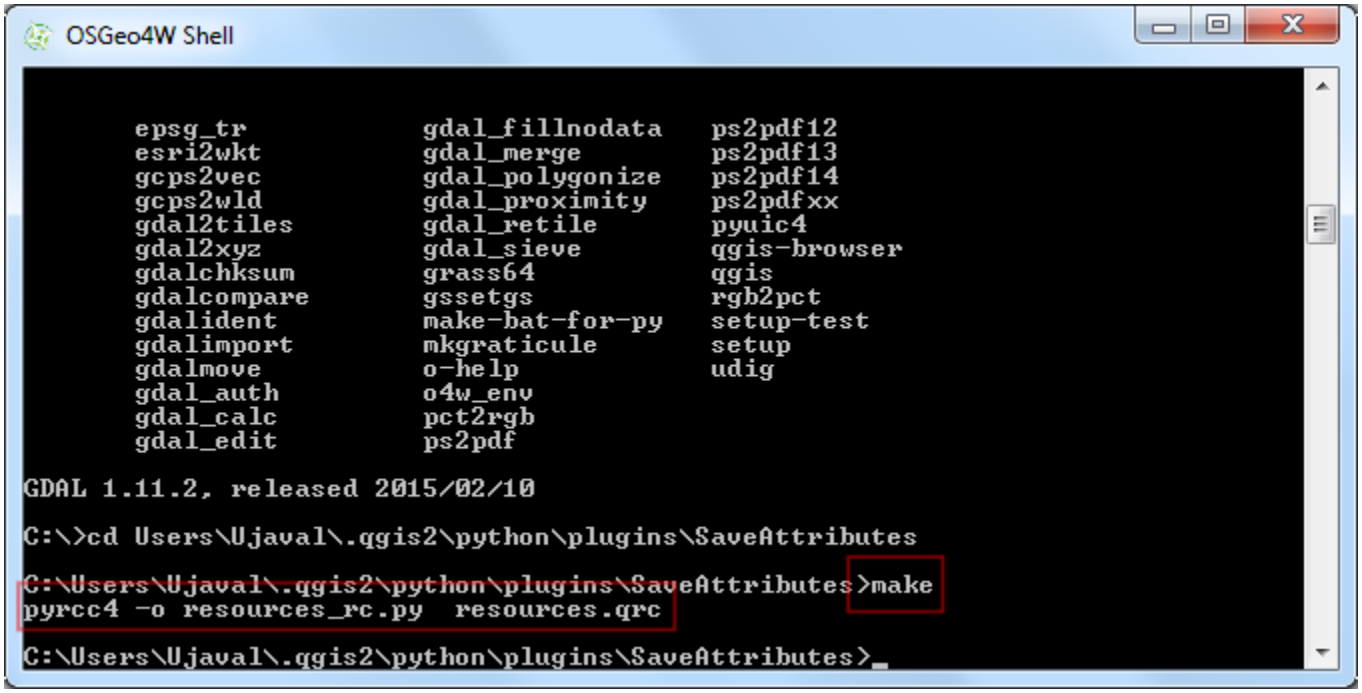
6. Browse to the plugin directory where the output of `Plugin Builder` was created. You can use the `cd` command followed by the path to the directory.

```
cd c:\Users\username\.qgis2\python\plugins\SaveAttributes
```

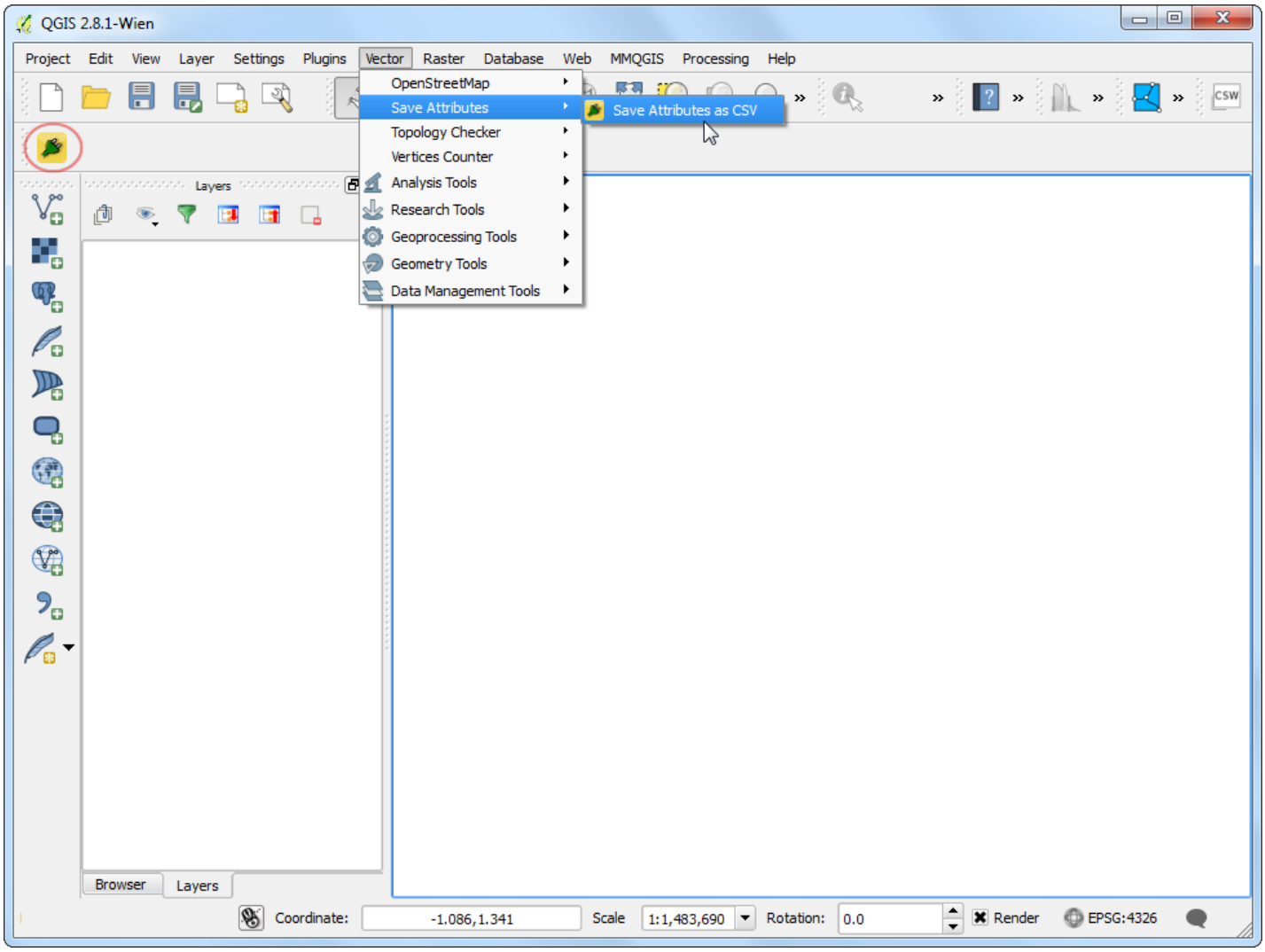


7. Once you are in the directory, type `make`. This will run the `pyrcc4` command that we had installed as part of Qt bindings for Python.

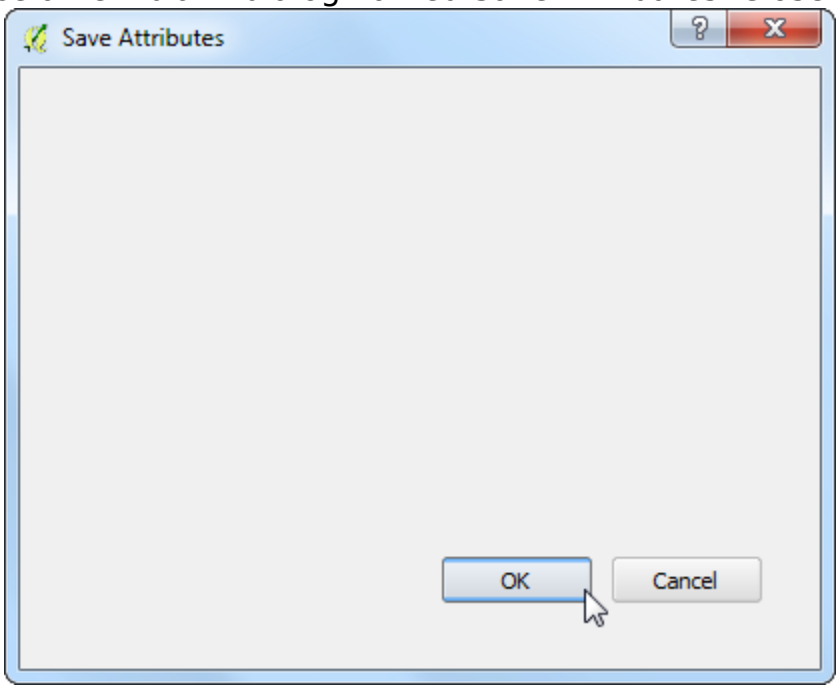
```
make
```



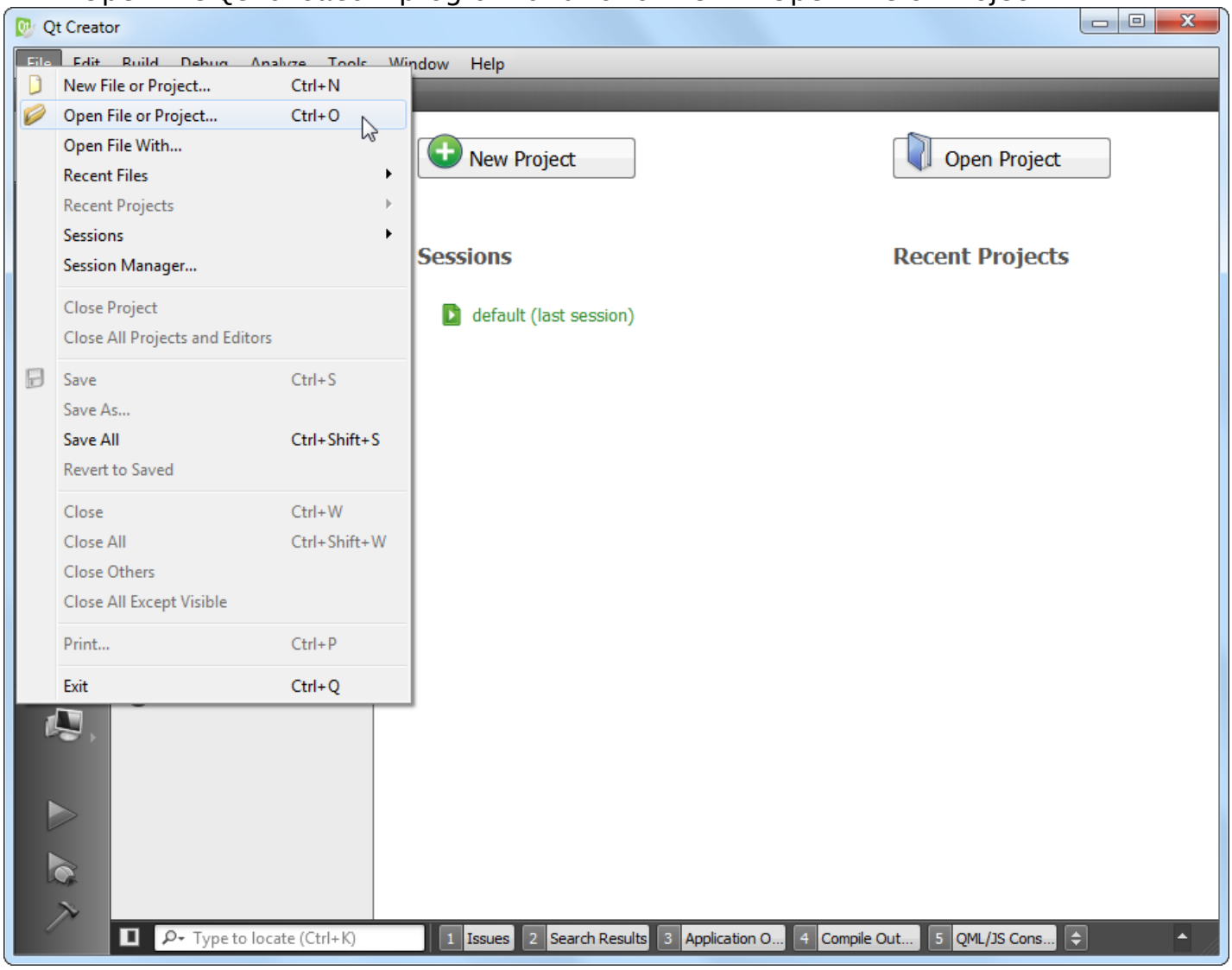
8. Now we are ready to have a first look at the brand new plugin we created. Close QGIS and launch it again. Go to Plugins ▸ Manage and Install plugins and enable the `Save Attributes` plugin in the Installed tab. You will notice that there is a new icon in the toolbar and a new menu entry under Vector ▸ Save Attributes ▸ Save Attributes as CSV`. Select it to launch the plugin dialog.



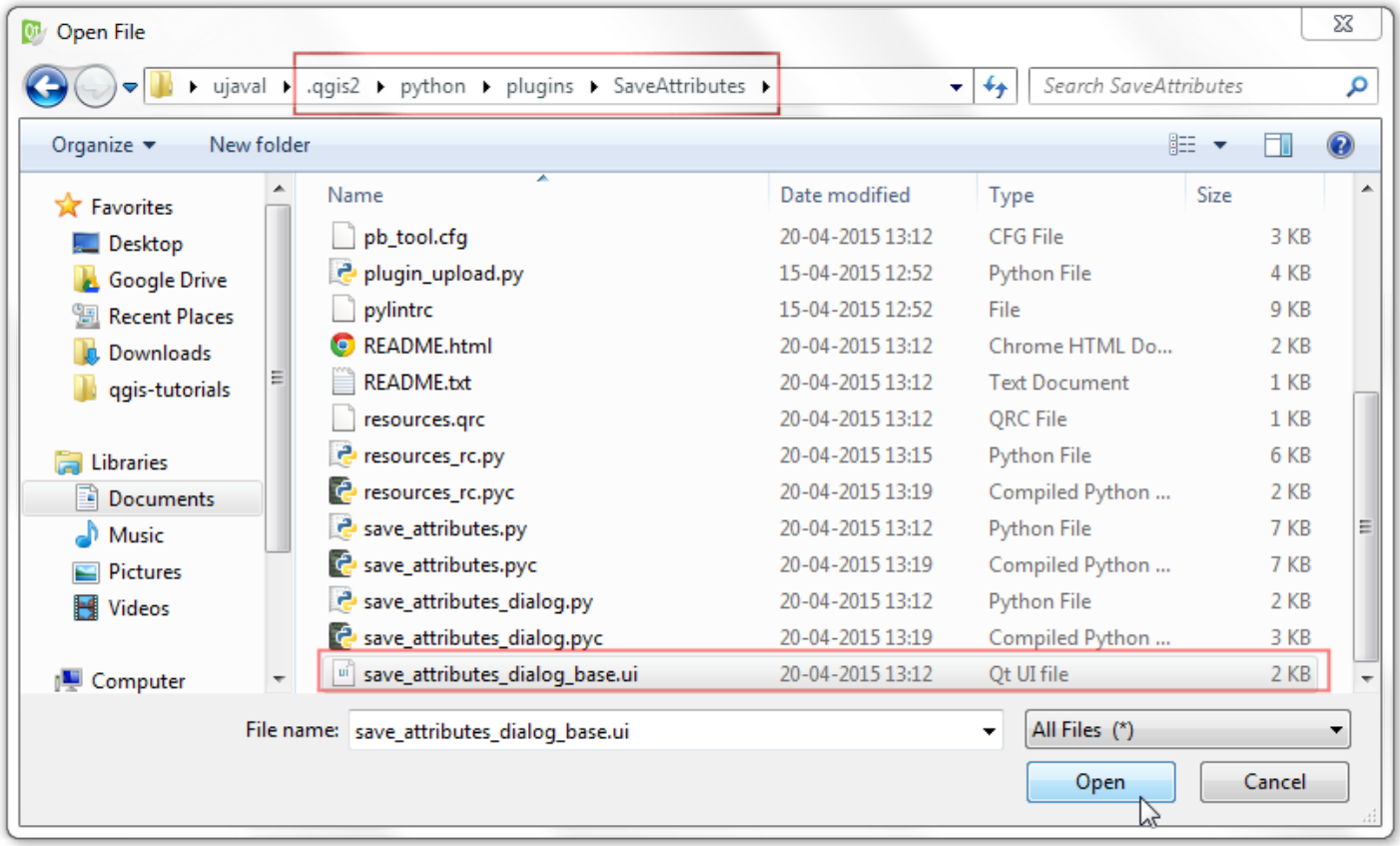
9. You will notice a new blank dialog named Save Attributes. Close this dialog.



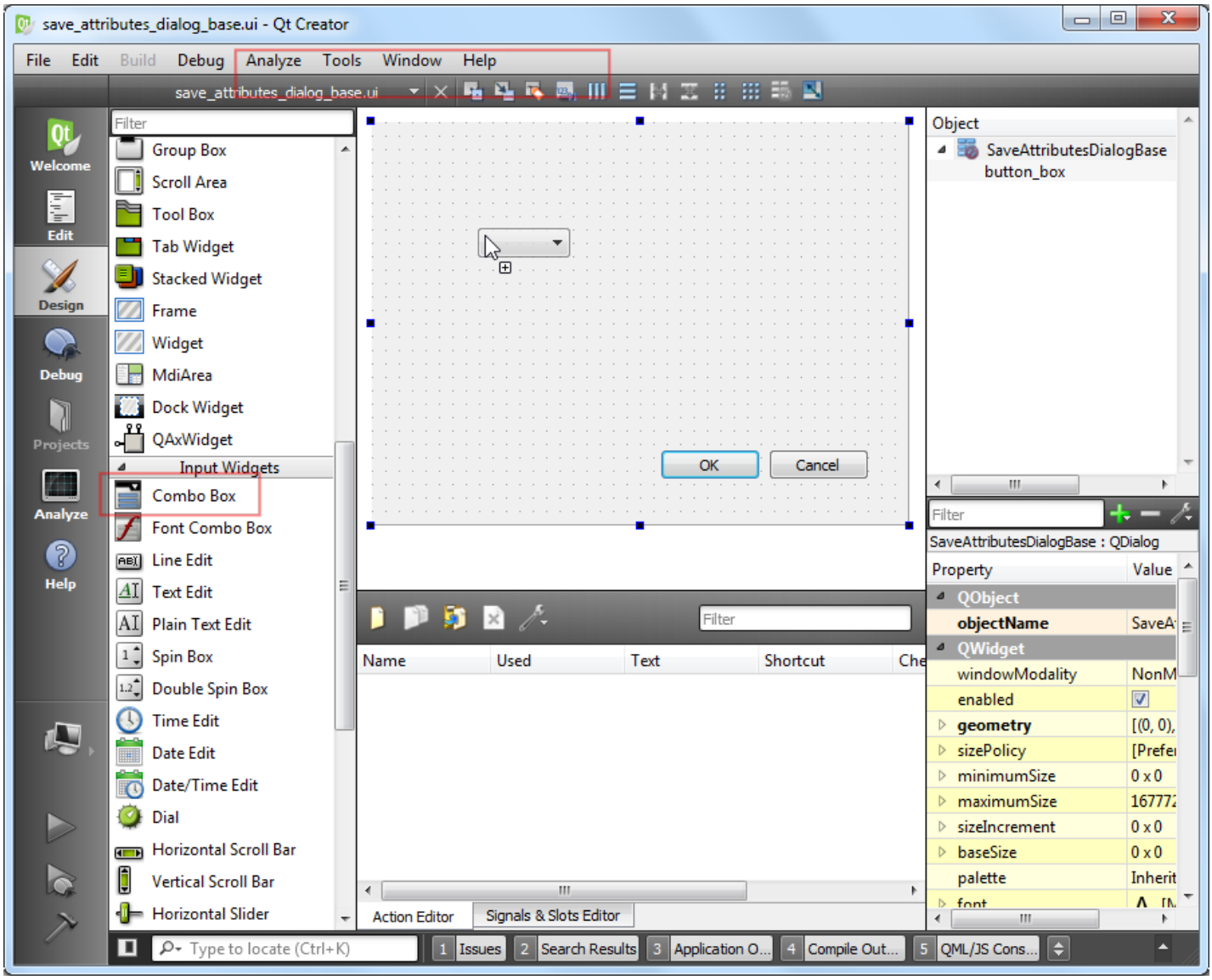
10. We will now design our dialog box and add some user interface elements to it. Open the `Qt Creator` program and to to File --> Open File or Project....



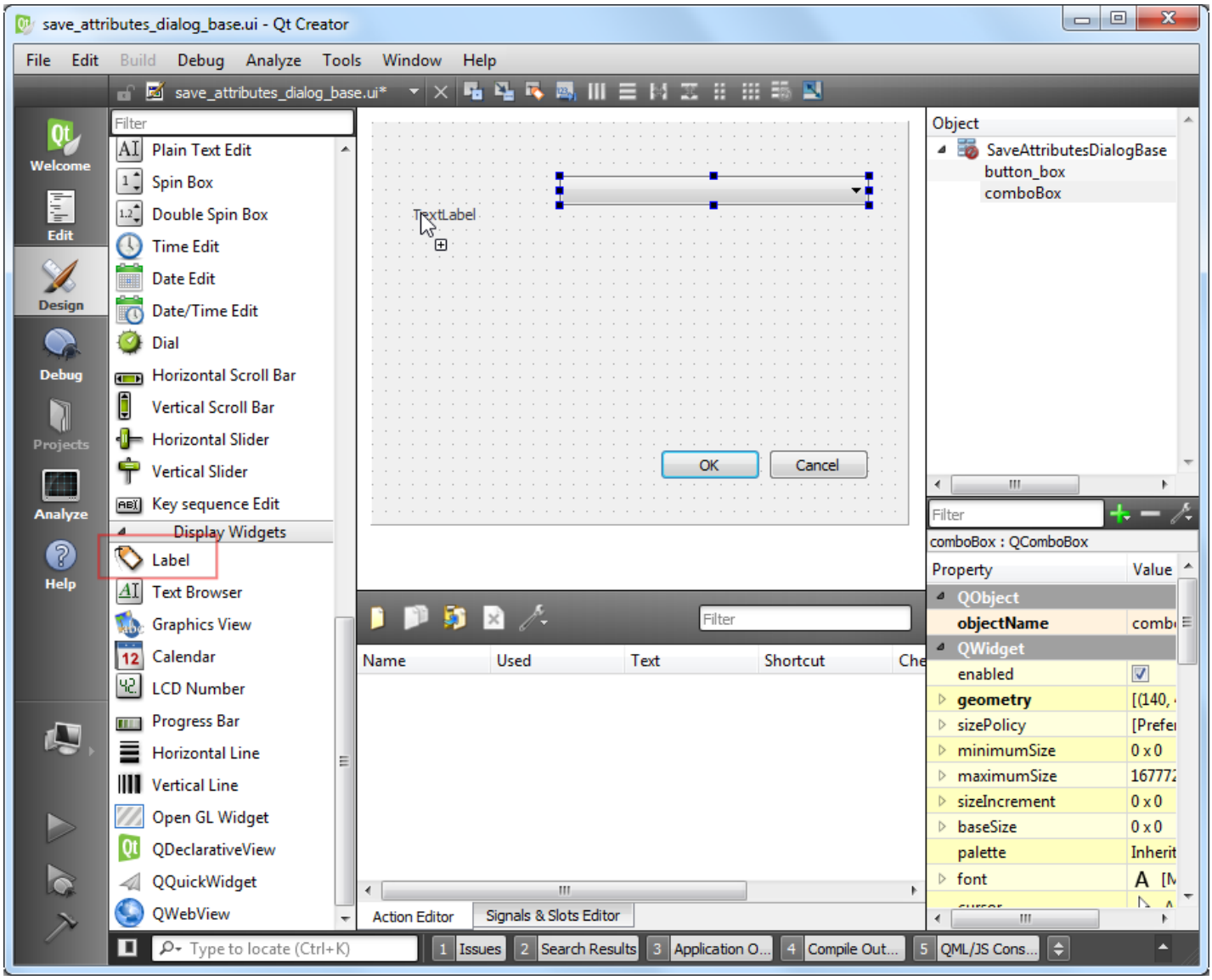
11. Browse to the plugin directory and select the `save_attributes_dialog_base.ui` file. Click Open.



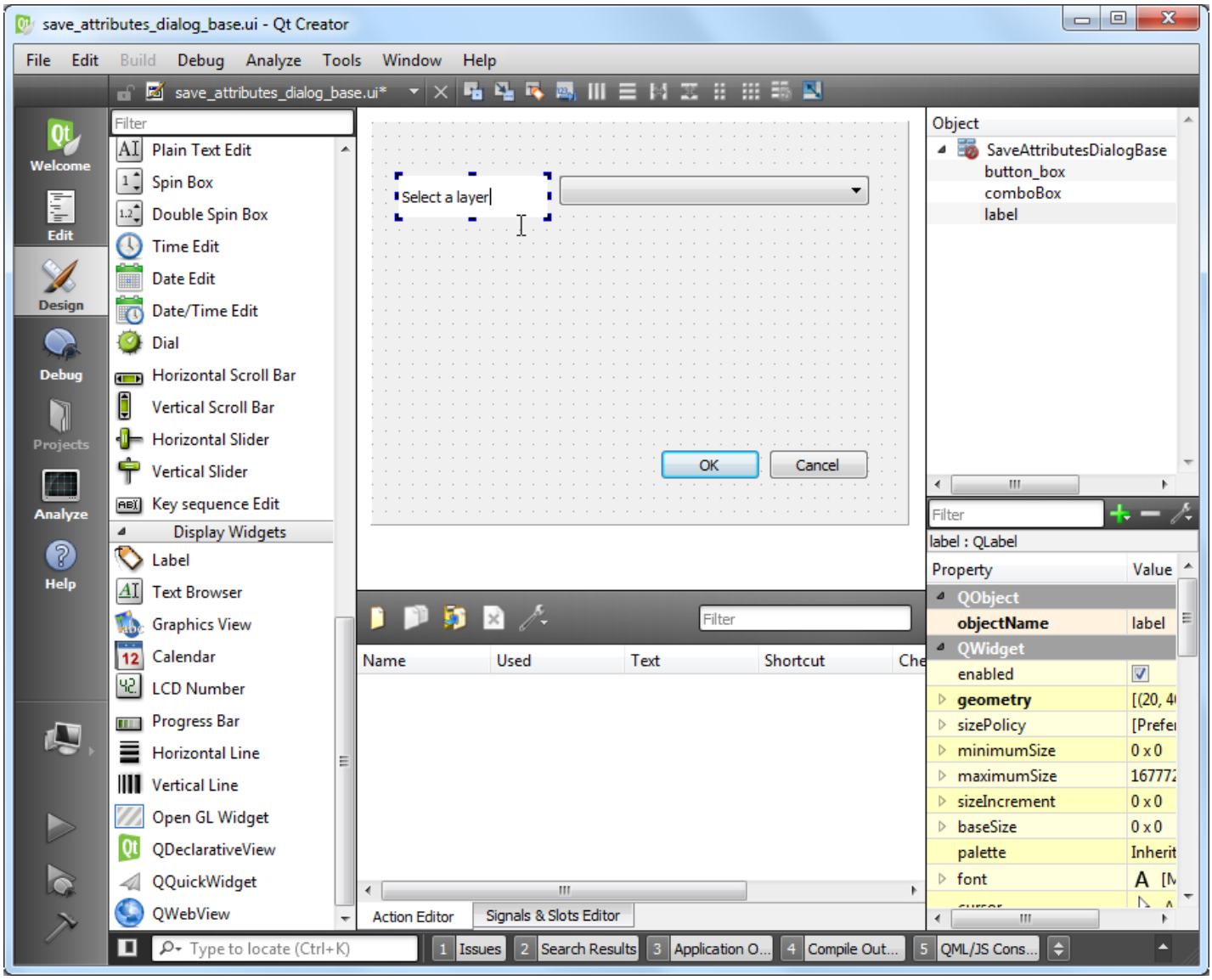
12. You will see the blank dialog from the plugin. You can drag-and-drop elements from the left-hand panel on the dialog. We will add a Combo Box type of Input Widget. Drag it to the plugin dialog.



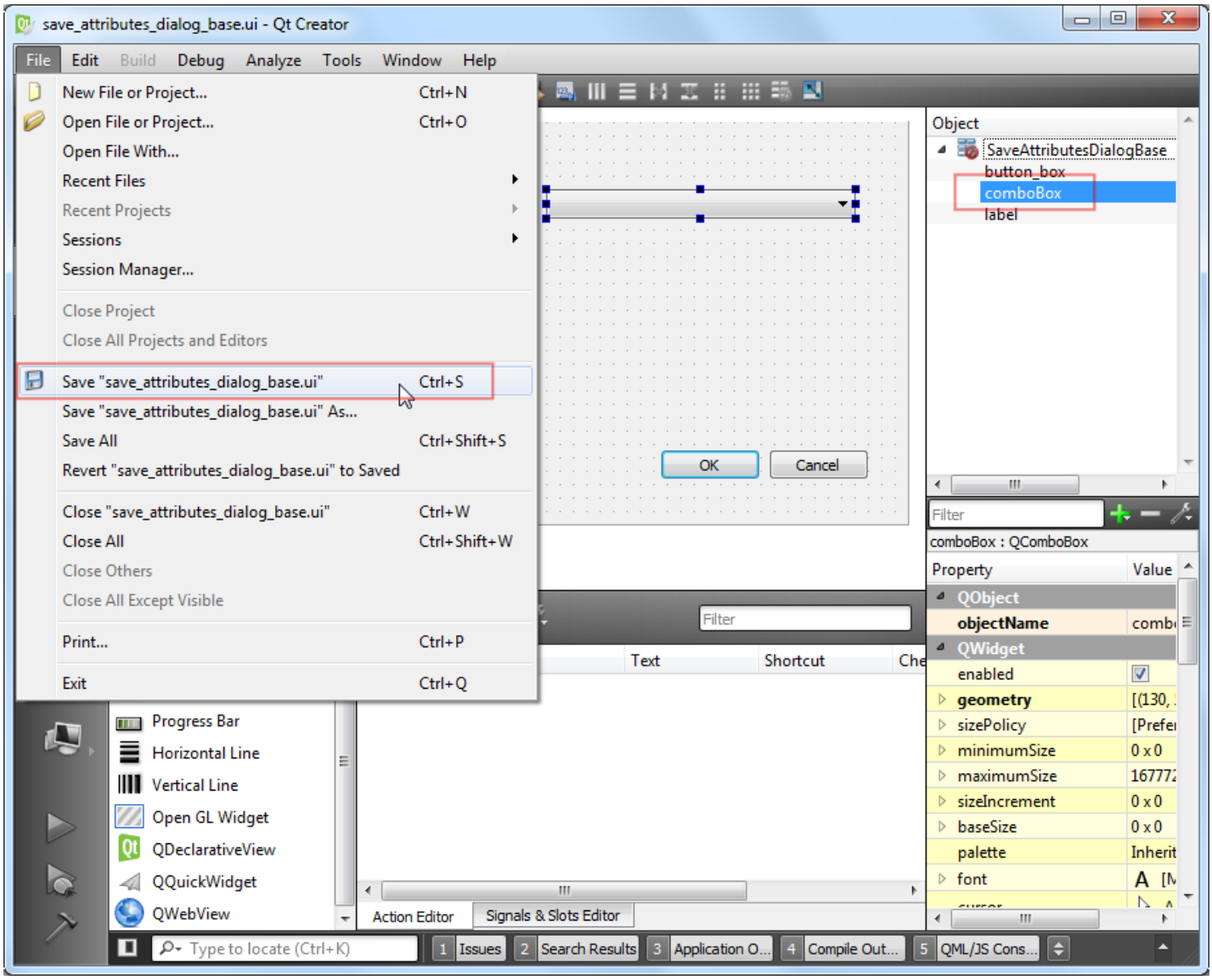
13. Resize the combo box and adjust its size. Now drag a Label type Display Widget on the dialog.



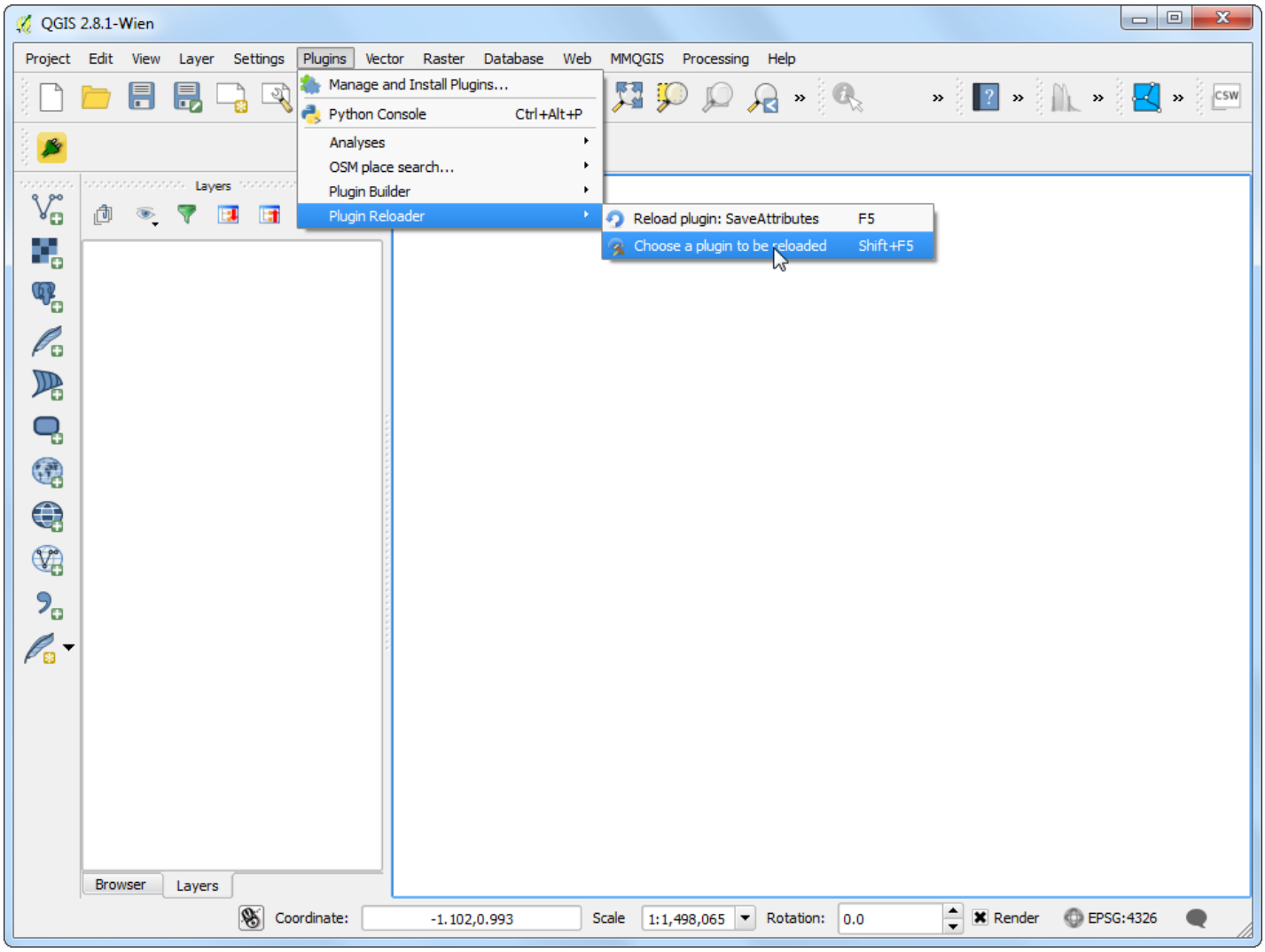
14. Click on the label text and enter `Select a layer`.



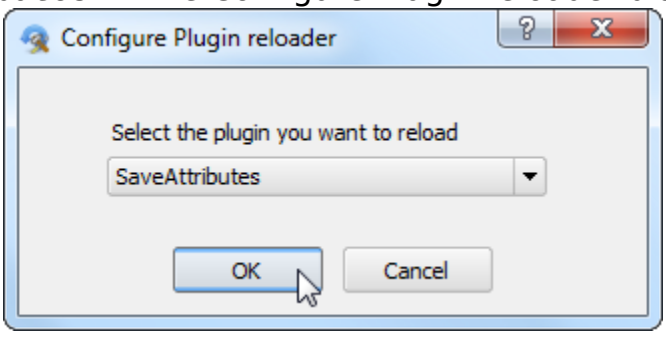
15. Save this file by going to File ▸ Save save_attributes_dialog_base.ui. Note the name of the combo box object is `comboBox`. To interact with this object using python code, we will have to refer to it by this name.



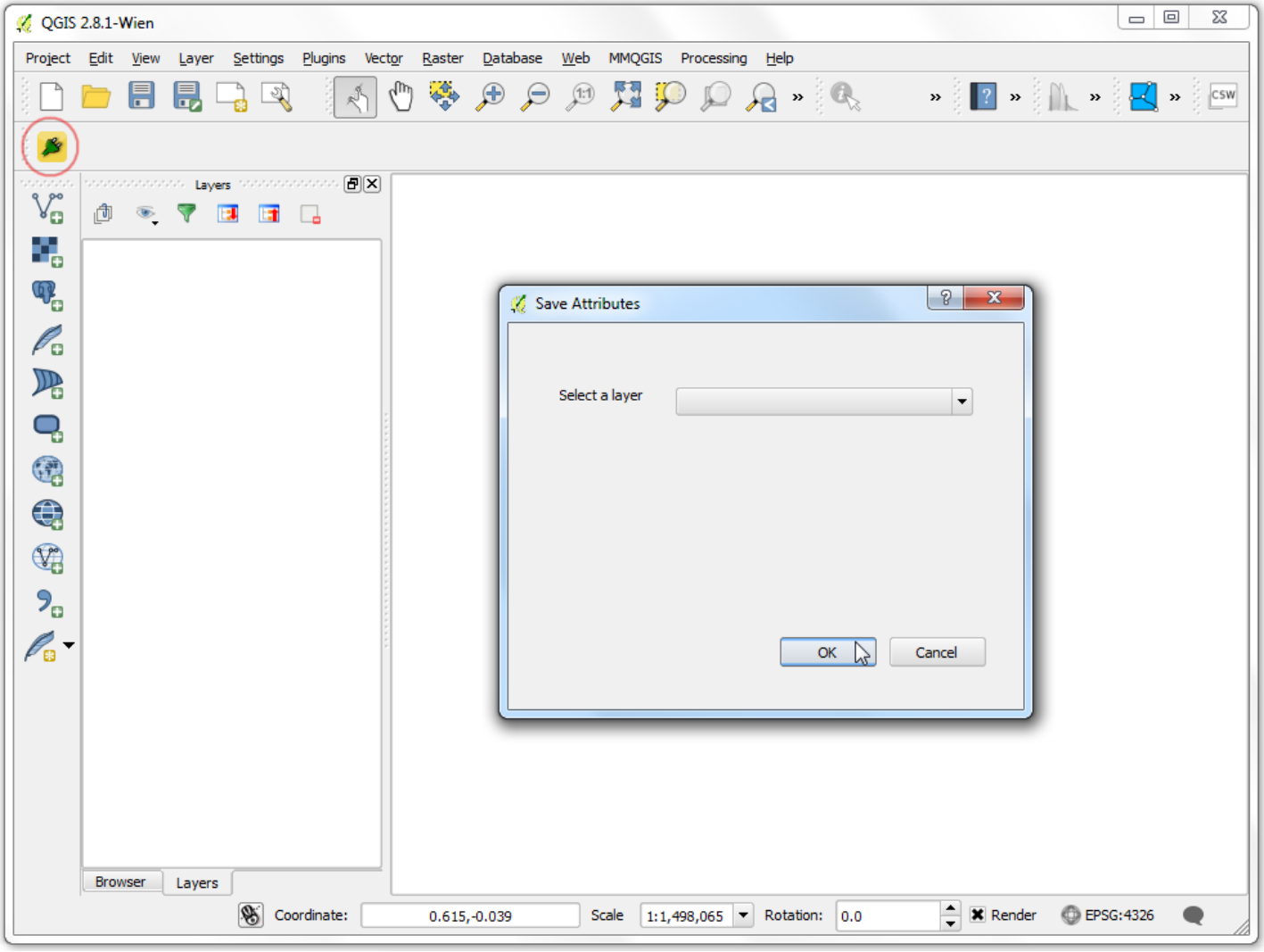
16. Let's reload our plugin so we can see the changes in the dialog window. Go to Plugin ▸ Plugin Reloader ▸ Choose a plugin to be reloaded.



17. Select `SaveAttributes` in the Configure Plugin reloader dialog.

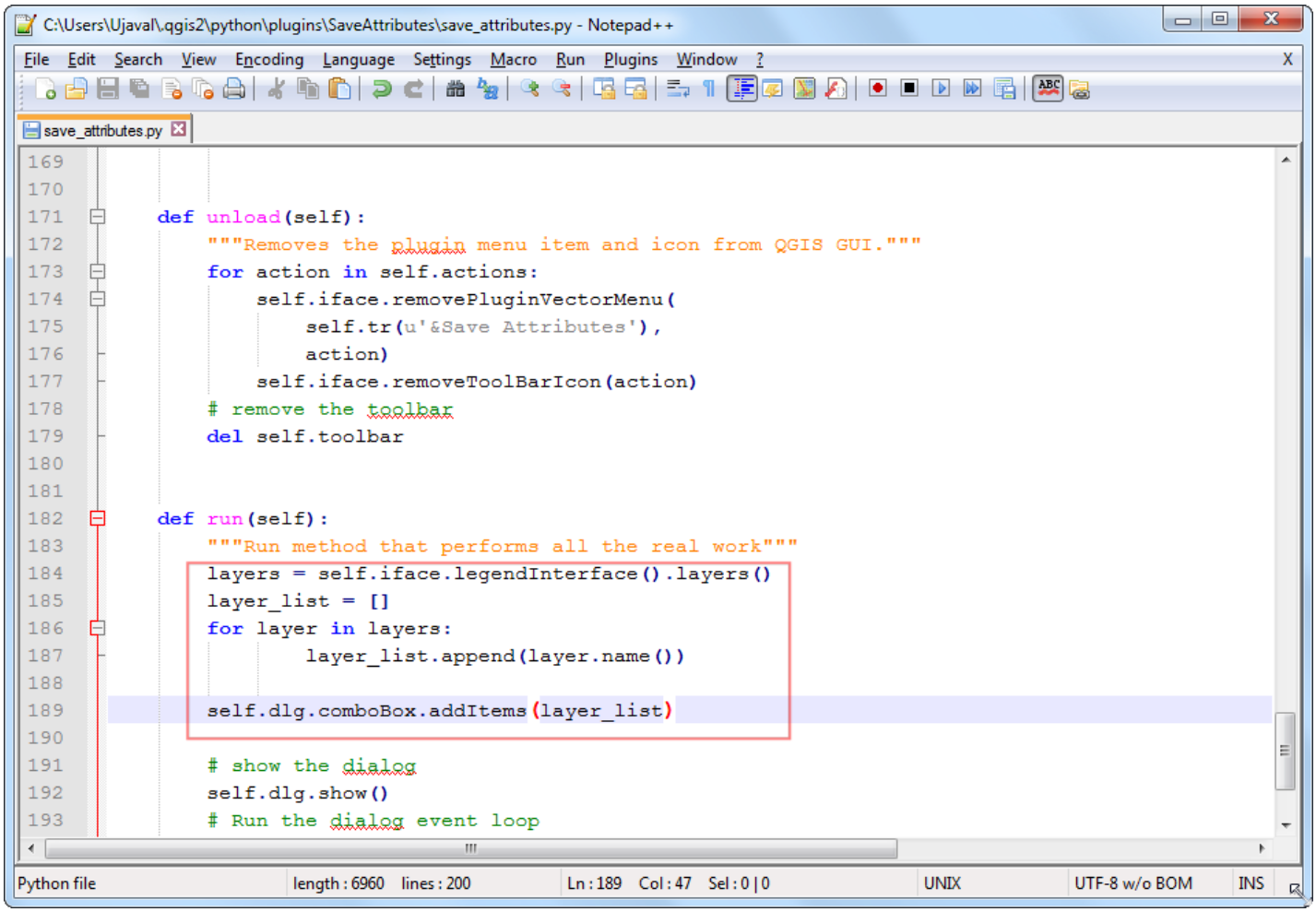


18. Now click the Save Attributes as CSV button. You will see the newly designed dialog box.

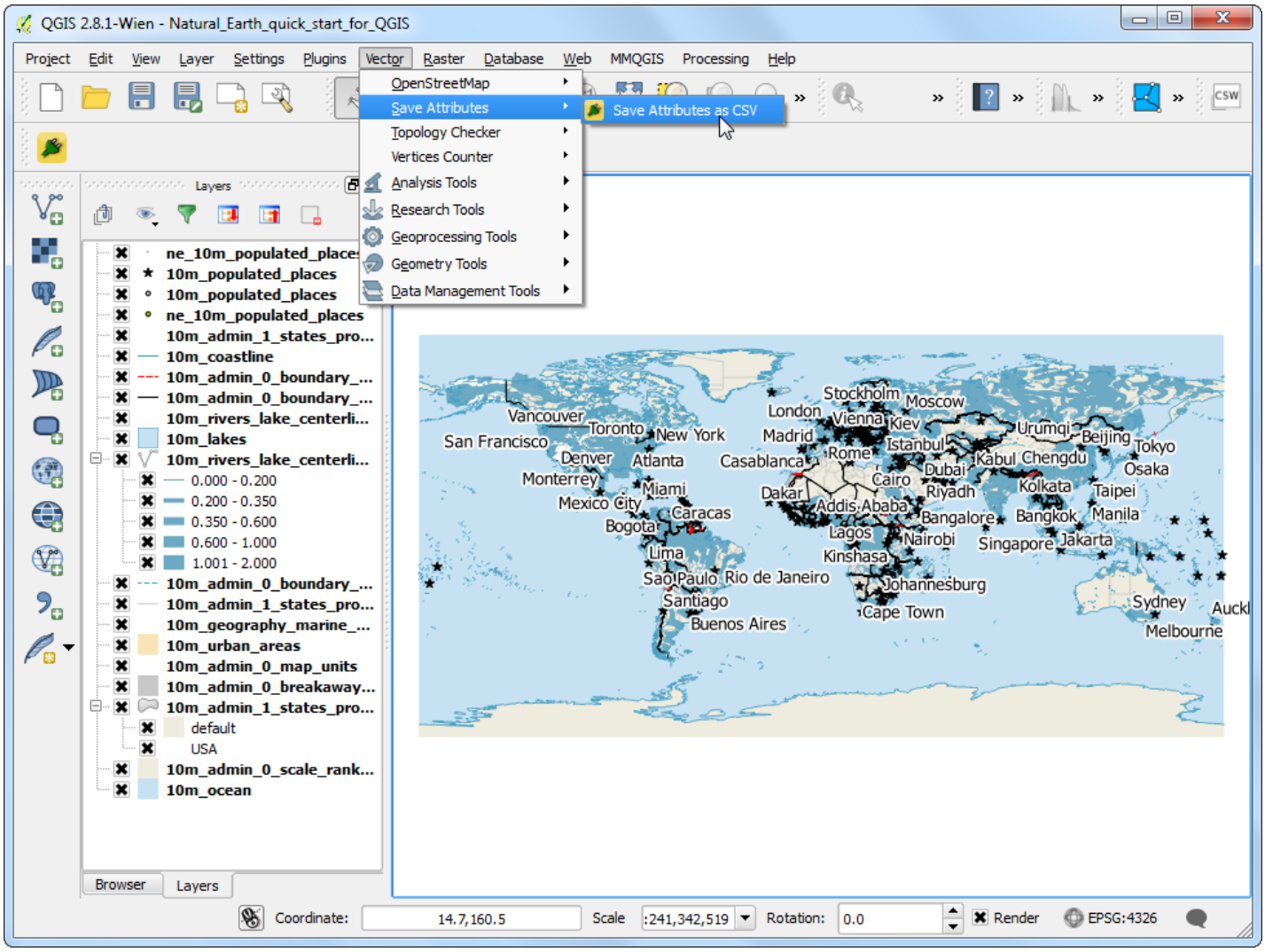


19. Let's add some logic to the plugin that will populate the combo box with the layers loaded in QGIS. Go to the plugin directory and load the file `save_attributes.py` in a text editor. Scroll down and find the `run(self)` method. This method will be called when you click the toolbar button or select the plugin menu item. Add the following code at the beginning of the method. This code gets the layers loaded in QGIS and adds it to the `comboBox` object from the plugin dialog.

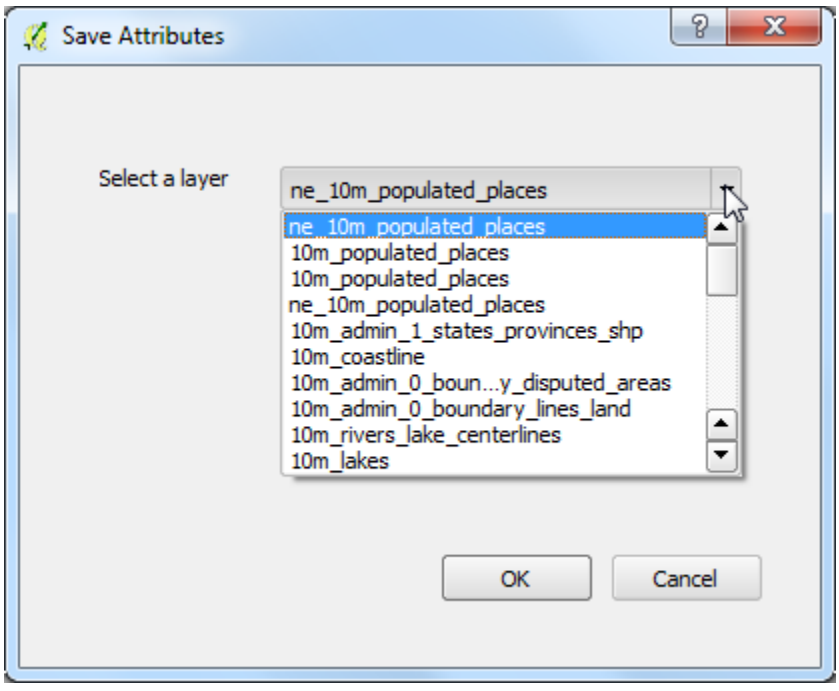
```
layers = self.iface.legendInterface().layers()
layer_list = []
for layer in layers:
     layer_list.append(layer.name())
     self.dlg.comboBox.addItems(layer_list)
```



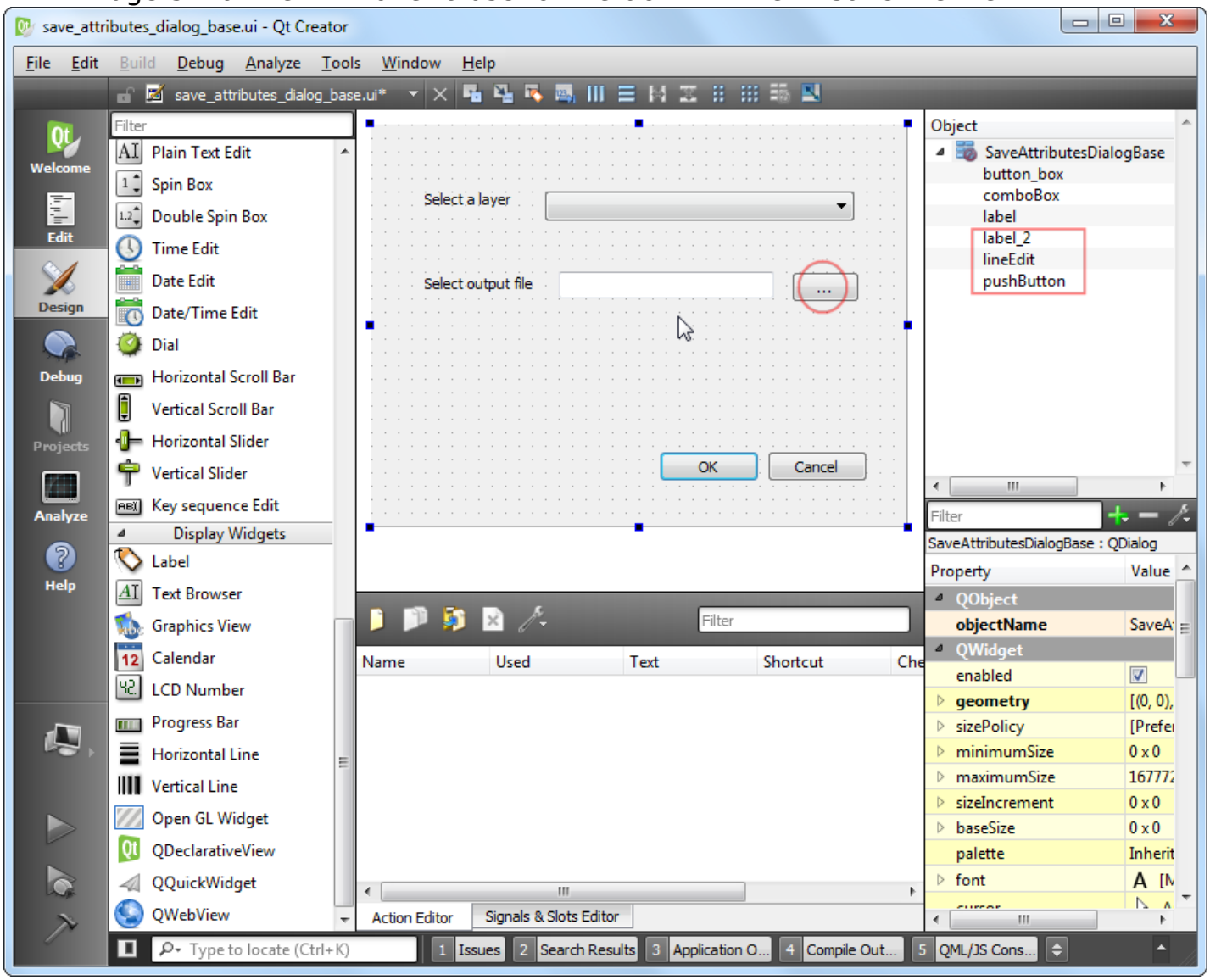
20. Back in the main QGIS window, reload the plugin by going to Plugins ▸ Plugin Reloader ▸ Reload plugin: SaveAttributes. Alternatively, you can just press F5. To test this new functionality, we must load some layers in QGIS. After you load some data, launch the plugin by going to Vector ▸ Save Attributes ▸ Save Attributes as CSV.



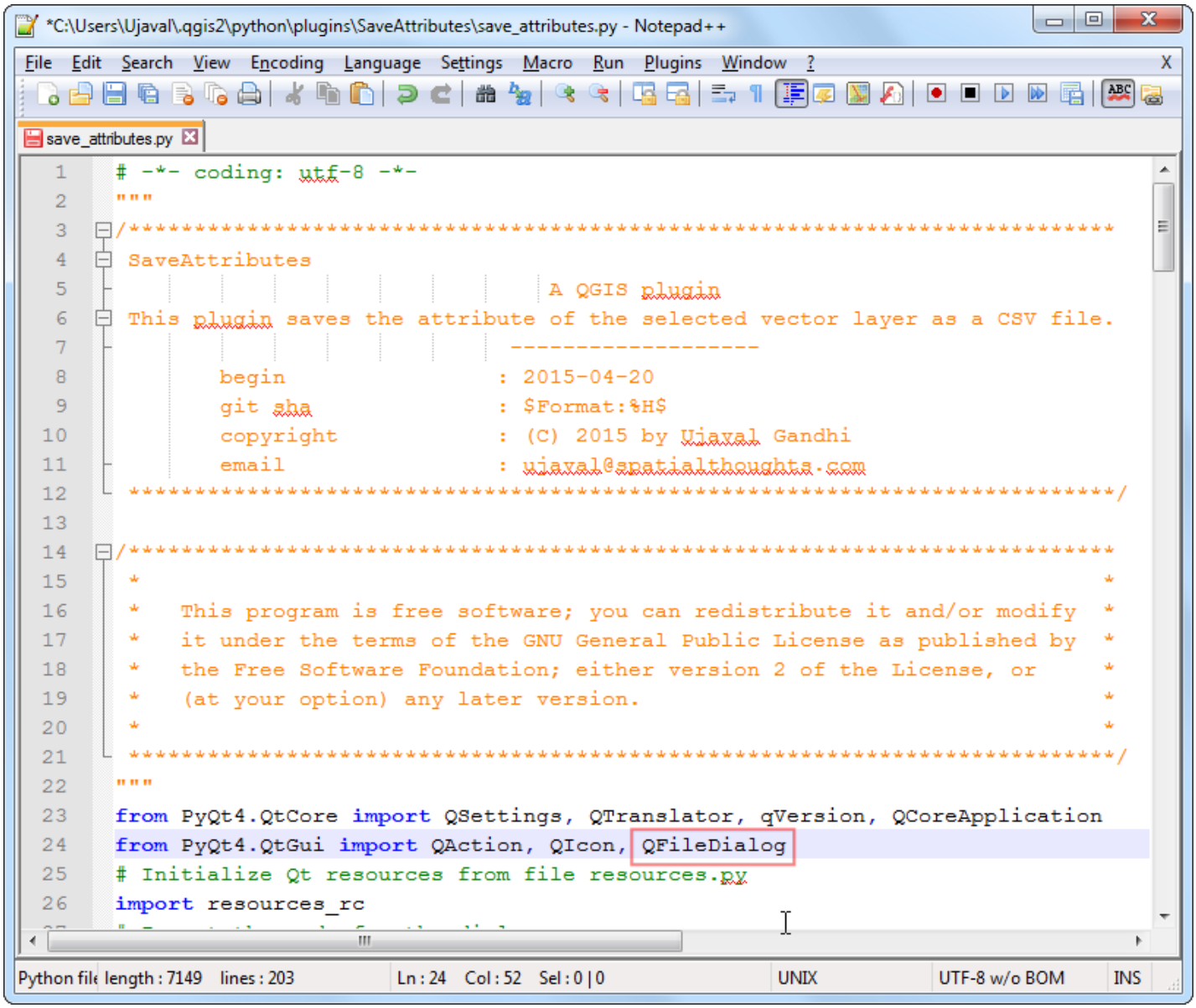
21. You will see that our combo box is now populated with the layer names that are loaded in QGIS.



22. Let's add remaining user interface elements. Switch back to `Qt Creator` and load the `save_attributes_dialog_base.ui` file. Add a `Label` Display Widget and change the text to `Select output file`. Add a `LineEdit` type Input Widget that will show the output file path that the user has chosen. Next, add a `Push Button` type Button and change the button label to `...`. Note the object names of the widgets that we will have to use to interact with them. Save the file.

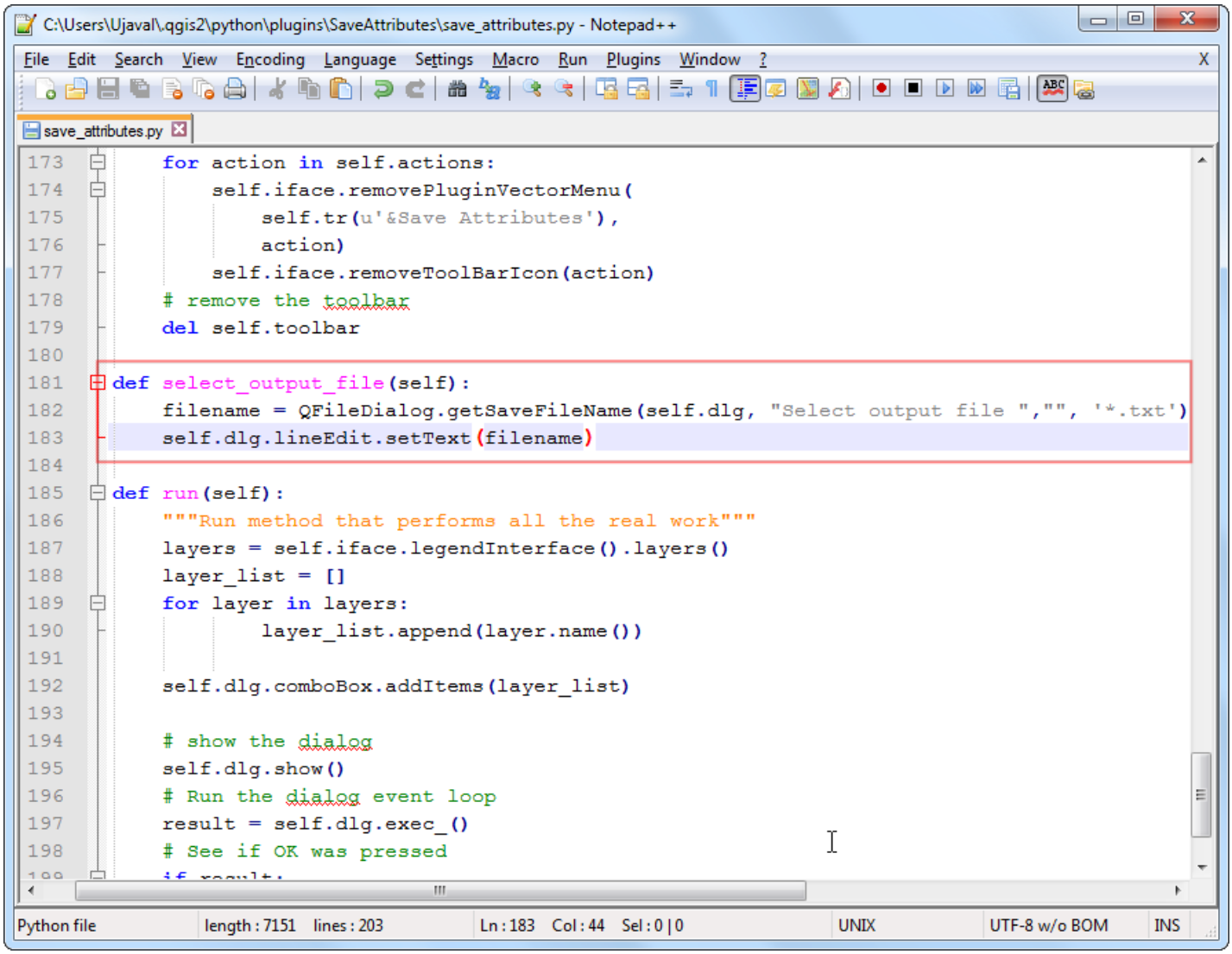


23. We will now add python code to open a file browser when the user clicks the `...` push button and show the select path in the line edit widget. Open the `save_attributes.py` file in a text editor. Add `QFileDialog` to our list of imports at the top of the file.



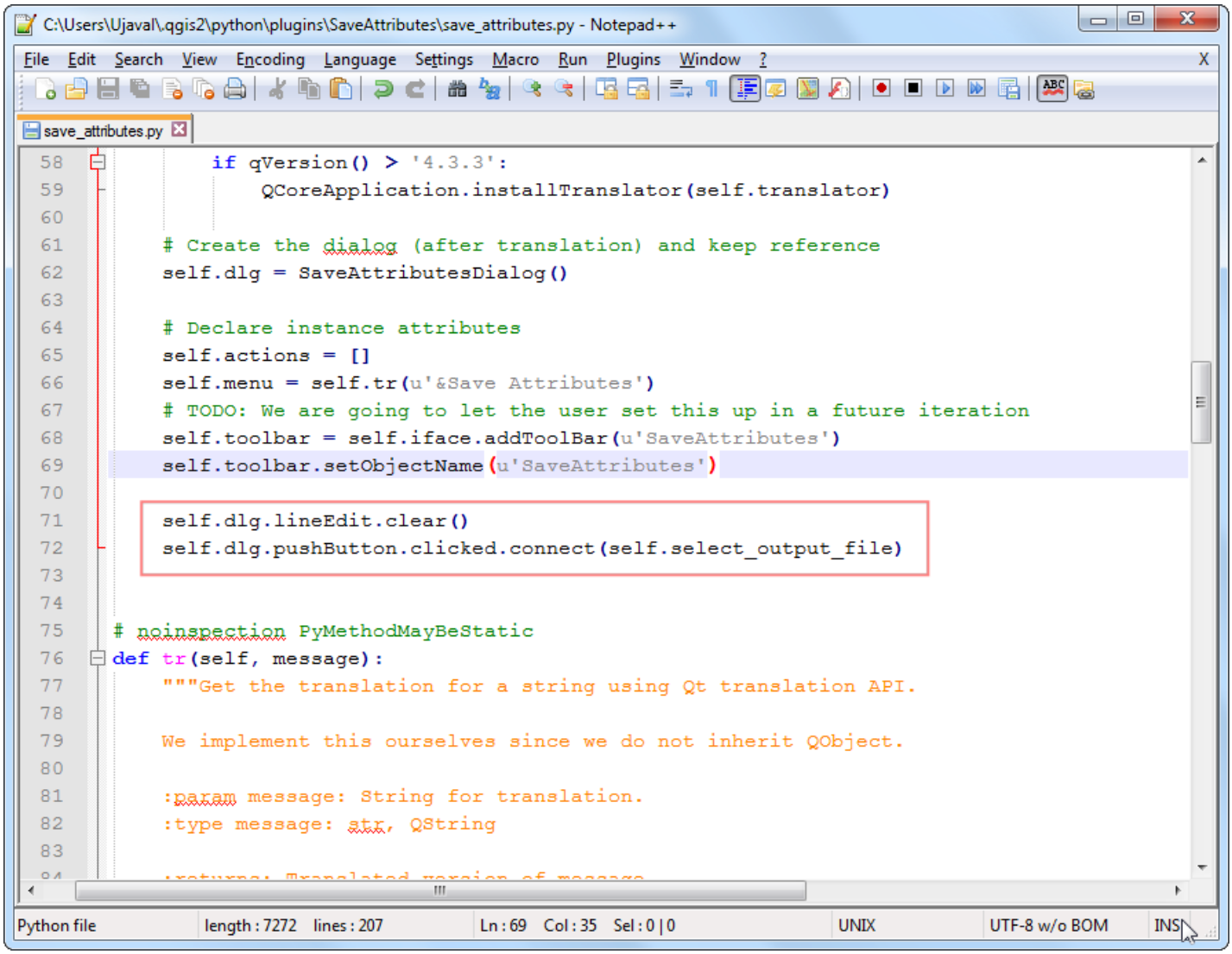
24. Add a new method called `select_output_file` with the following code. This code will open a file browser and populate the line edit widget with the path of the file that the user chose.

```
def select_output_file(self):
    filename = QFileDialog.getSaveFileName(self.dlg, "Select output file ","", '*.txt')
    self.dlg.lineEdit.setText(filename)
```

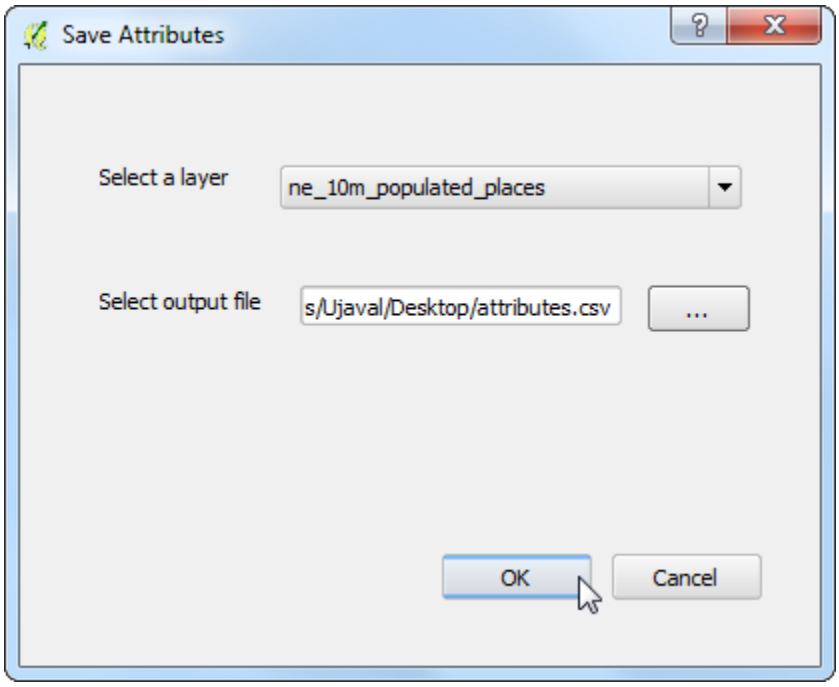


25. Now we need to add code so that when the … button is clicked, `select_output_file` method is called. Scroll up to the `__init__` method and add the following lines at the bottom. This code will clear the previously loaded text (if any) in the line edit widget and also connect the `select_output_file` method to the `clicked` signal of the push button widget.

```
self.dlg.lineEdit.clear()
self.dlg.pushButton.clicked.connect(self.select_output_file)
```



26. Back in QGIS, reload the plugin and open the Save Attributes` dialog. If all went fine, you will be able to click the . . . button and select an output text file from your disk.

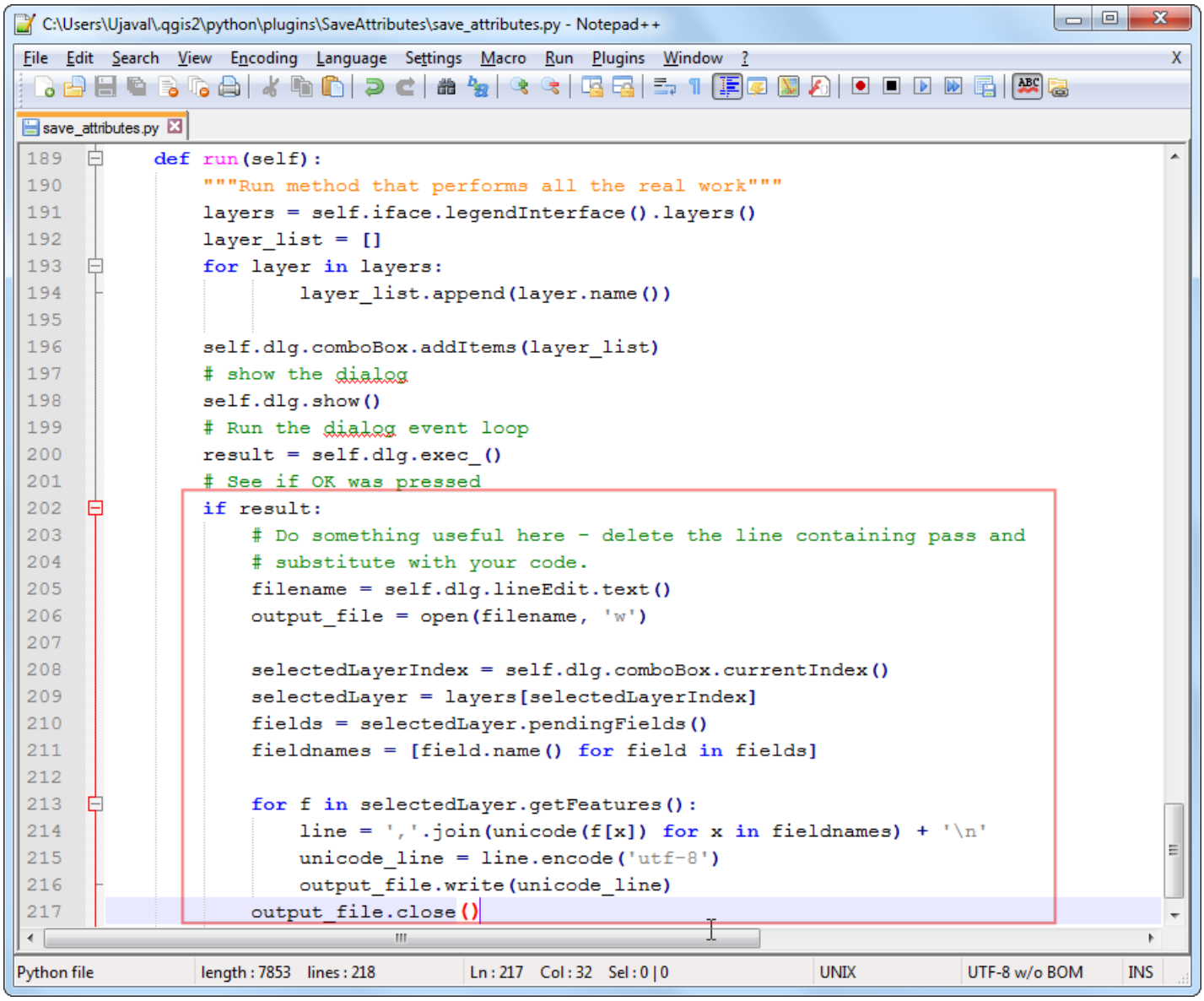


27. When you click OK on the plugin dialog, nothing happens. That is because we have not added the logic to pull attribute information from the layer and write it to the text file. We now have all the pieces in place to do just that. Find the place in the `run` method where it says `pass`. Replace it with the code below. The explanation for this code can be found in *Getting Started With Python Programming*.

```
filename = self.dlg.lineEdit.text()
output_file = open(filename, 'w')

selectedLayerIndex = self.dlg.comboBox.currentIndex()
selectedLayer = layers[selectedLayerIndex]
fields = selectedLayer.pendingFields()
fieldnames = [field.name() for field in fields]

for f in selectedLayer.getFeatures():
    line = ','.join(unicode(f[x]) for x in fieldnames) + '\n'
    unicode_line = line.encode('utf-8')
    output_file.write(unicode_line)
output_file.close()
```



28. Now our plugin is ready. Reload the plugin and try it out. You will find that the output text file you chose will have the attributes from the vector layer. You can zip the plugin directory and share it with your users. They can unzip the contents to their plugin directory and try out your plugin. If this was a real plugin, you would upload it to the QGIS Plugin Repository so that all QGIS users will be able to find and download your plugin.

Note

This plugin is for demonstration purpose only. Do not publish this plugin or upload it to the QGIS plugin repository.

Below is the full `save_attributes.py` file as a reference.

```
# -*- coding: utf-8 -*-
"""
```

```
/***************************************************************************
 SaveAttributes
                                 A QGIS plugin
 This plugin saves the attribute of the selected vector layer as a CSV file.
                              -------------------
        begin                : 2015-04-20
        git sha              : $Format:%H$
        copyright            : (C) 2015 by Ujaval Gandhi
        email                : ujaval@spatialthoughts.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
from PyQt4.QtGui import QAction, QIcon, QFileDialog
# Initialize Qt resources from file resources.py
import resources_rc
# Import the code for the dialog
from save_attributes_dialog import SaveAttributesDialog
import os.path


class SaveAttributes:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'SaveAttributes_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)
```

```
        # Create the dialog (after translation) and keep reference
        self.dlg = SaveAttributesDialog()

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&Save Attributes')
        # TODO: We are going to let the user set this up in a future iteration
        self.toolbar = self.iface.addToolBar(u'SaveAttributes')
        self.toolbar.setObjectName(u'SaveAttributes')

        self.dlg.lineEdit.clear()
        self.dlg.pushButton.clicked.connect(self.select_output_file)


    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('SaveAttributes', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.

        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this action.
        :type text: str

        :param callback: Function to be called when the action is triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be enabled
            by default. Defaults to True.
        :type enabled_flag: bool
```

```
        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is also
            added to self.actions list.
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            self.toolbar.addAction(action)

        if add_to_menu:
            self.iface.addPluginToVectorMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/SaveAttributes/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'Save Attributes as CSV'),
            callback=self.run,
            parent=self.iface.mainWindow())
```

```
    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginVectorMenu(
                self.tr(u'&Save Attributes'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar

    def select_output_file(self):
        filename = QFileDialog.getSaveFileName(self.dlg, "Select output file ","", '*.txt')
        self.dlg.lineEdit.setText(filename)

    def run(self):
        """Run method that performs all the real work"""
        layers = self.iface.legendInterface().layers()
        layer_list = []
        for layer in layers:
                layer_list.append(layer.name())

        self.dlg.comboBox.clear()
        self.dlg.comboBox.addItems(layer_list)
        # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            # Do something useful here - delete the line containing pass and
            # substitute with your code.
            filename = self.dlg.lineEdit.text()
            output_file = open(filename, 'w')

            selectedLayerIndex = self.dlg.comboBox.currentIndex()
            selectedLayer = layers[selectedLayerIndex]
            fields = selectedLayer.pendingFields()
            fieldnames = [field.name() for field in fields]

            for f in selectedLayer.getFeatures():
                line = ','.join(unicode(f[x]) for x in fieldnames) + '\n'
                unicode_line = line.encode('utf-8')
                output_file.write(unicode_line)
            output_file.close()
```