

## **BUILDING QGIS PLUGINS**

QGIS has made provision to extend its functionality by writing Plugins in Python and C++. Plugins are categorized as core, stable and experimental. Several Plugins are quite extensively used such as “OpenLayers” and “Semi-Automatic Classification Plugin”. 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.

This tutorial is a derivative of the Building a Python Plugin Tutorial<sup>1</sup> by Ujal Gandhi.

**Purpose:** To build a clip tool which updates the geometry columns

**Current Limitation:** The existing clip tool does not update the geometry columns

**Pre-requisites:** Knowledge of GIS data is a must. Working knowledge of Python programming will be added advantage.

**QGIS Version:** 3.6.0-Noosa +

**Additional Software required:** (1) Qt Designer (comes bundled with the standard QGIS installation), (2) QGIS Plugin - Plugin Builder, (3) QGIS Plugin - Reloader plugin and (4) a beautiful editor - Sublime Text 2

*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.*

We shall try to clip Snow cover area of North India with the district boundary of Uttarkashi, Uttarakhand and notice the change in the geometry columns.

*For the ease of understanding, both the layers used are in the UTM 44N / WGS 84 “projected” coordinate system. The map units are in meters.*

1. Add 2 overlapping vector layers. Click on Layer - Add Layer - Add Vector Layer. Navigate to the folder shared with you, press Ctrl and select the two files “Snow\_NorthIndia.shp” and “Uttarkashi.shp” (Fig 1)
2. Click on Vector - Geometry tools - Add / Export Geometry columns. Select “Snow\_NorthIndia” in the drop and click Ok (Fig 2). Close the box, after process is over.
3. Now right click on the new layer “Added geom info”, click on Open Attribute Table. Note the values of AREA and PERIMETER columns (esp., for features with POLY\_ID = 79, 80). These are the area and perimeter values before the clip operation. (Fig 3)
4. Now perform the clip operation and save the output file as “clip\_before.shp”. Open the Attribute table of the “clip\_before.shp” file and compare the area and perimeter values noted down in the previous step. You will notice that although the features have changed, the area and perimeter values have not been updated.

---

<sup>1</sup> [http://www.qgistutorials.com/en/docs/3/building\\_a\\_python\\_plugin.html](http://www.qgistutorials.com/en/docs/3/building_a_python_plugin.html)

**Figure 1:** QGIS interface showing the 'Layer' menu open with 'Add Vector Layer...' highlighted.

**Figure 2:** QGIS interface showing the 'Vector' menu open with 'Add Geometry Attributes...' highlighted.

**Figure 3:** QGIS attribute table showing two selected features (DN 79 and DN 80) highlighted with red boxes.

**Figure 4:** QGIS interface showing the 'Vector' menu open with 'Clip...' highlighted.

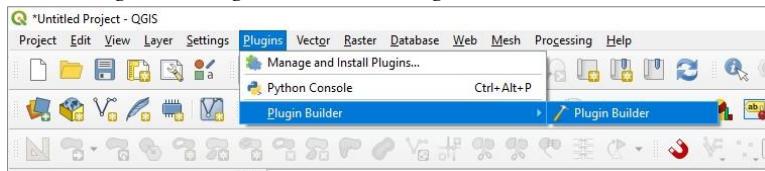
**Figure 5:** QGIS attribute table showing the result of the clipping operation, with the first feature (DN 1) highlighted with a green box.

Figures 1-5 (top left to bottom right in row-wise manner)

Now we shall try to create a tool to clip Snow cover area of North India with the district boundary of Uttarkashi, Uttarakhand and update the geometry columns also.

## Procedure

1. Open QGIS. Go to *Plugins - Plugin Builder - Plugin Builder*.



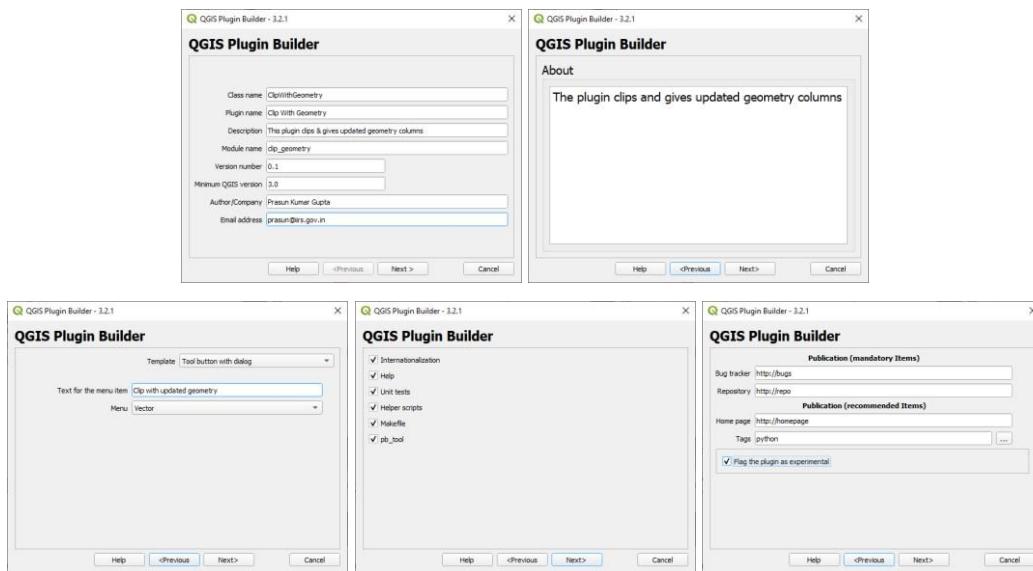
You will see the *QGIS Plugin Builder* dialog with a form. You can fill the form with details relating to our plugin. The *Class name* will be the name of the Python Class containing the logic of the plugin. This will also be the name of the folder containing all the plugin files. Enter `ClipWithGeometry` as the class name. The *Plugin name* is the name under which your plugin will appear in the *Plugin Manager*. Enter the name as “Clip With Geometry”. 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 `clip_geometry`. Leave the version numbers as they are. Enter your name and email address in the appropriate fields. Click on *Next*.

Enter some text in the “About” box and click on *Next*.

Select the “Tool button with dialog” from the Template selector. The *Text for menu item* value will be how the users will find your plugin in QGIS menu. Enter it as “Clip with Update Geometry”. The *Menu* field will decide where your plugin item is added in QGIS. Since our plugin is for vector data, select *Vector*. Click on *Next*.

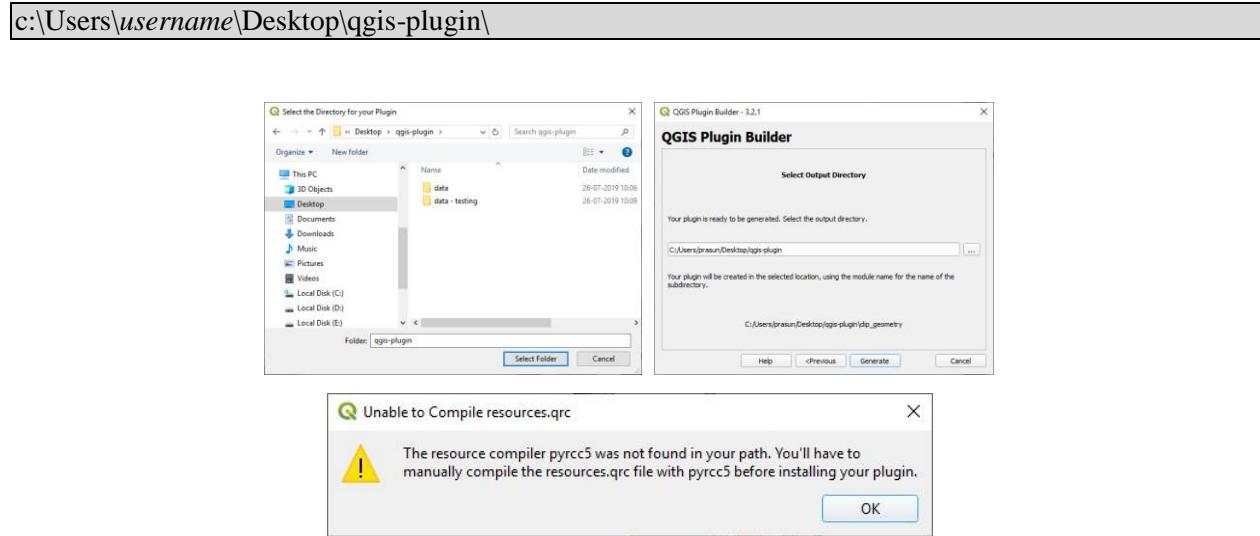
Plugin builder will prompt you for the type of files to generate. Keep the default selection and click *Next*.

Check the *Flag the plugin as experimental* box at the bottom. Click *Next*.

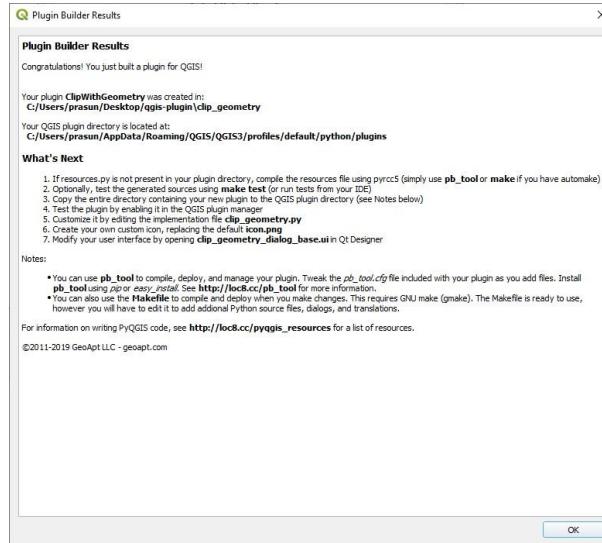


3. Next, you will be prompted to choose a directory for your plugin. You need to browse to the QGIS python plugin directory on your computer and select *Select Folder*. Provide the path of today's folder: (Replace username with your login name). Click on Generate. You may get a warning saying pyrcc5 is not found in the path. You can ignore the message.

## Windows

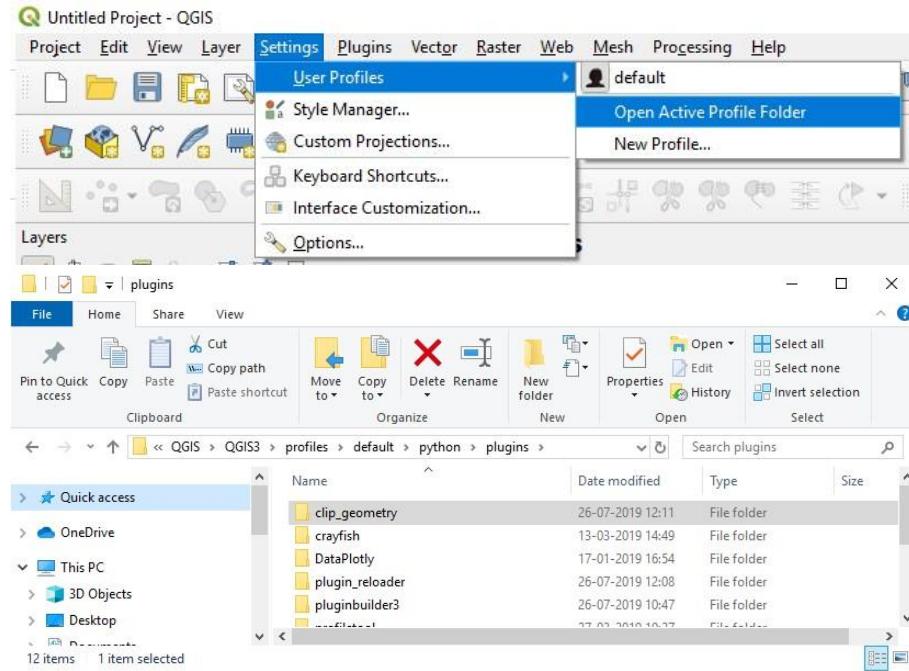


4. You will see a confirmation dialog once your plugin template is created. Note the path to the plugin folder.

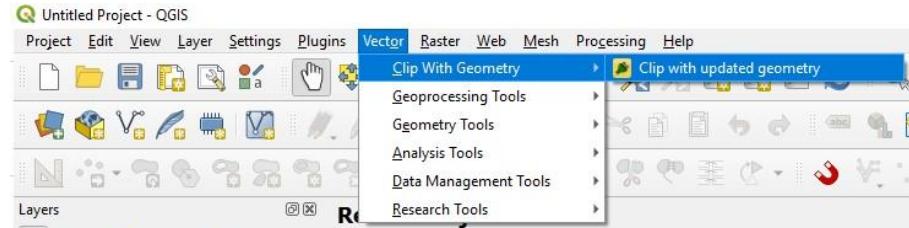


**THIS PORTION HAS BEEN RETRACTED**

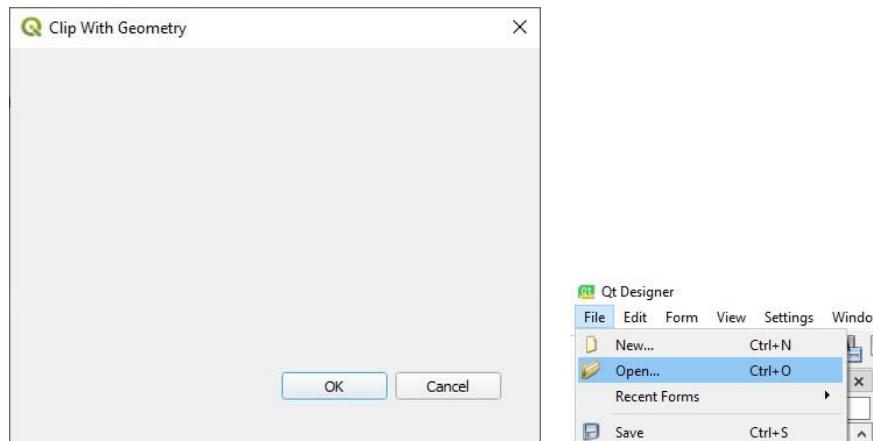
8. Plugins in QGIS are stored in a special folder. We must copy our plugin directory to that folder before it can be used. In QGIS, locate your current profile folder by going to Settings ▶ User Profiles ▶ Open Active Profile Folder. In the profile folder, copy the plugin folder to python ▶ plugins subfolder.



9. 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 “Clip With Geometry” plugin in the *Installed* tab. You will notice that there is a new icon in the toolbar and a new menu entry under *Vector - Clip With Geometry - Clip with Update Geometry*. Select it to launch the plugin dialog.

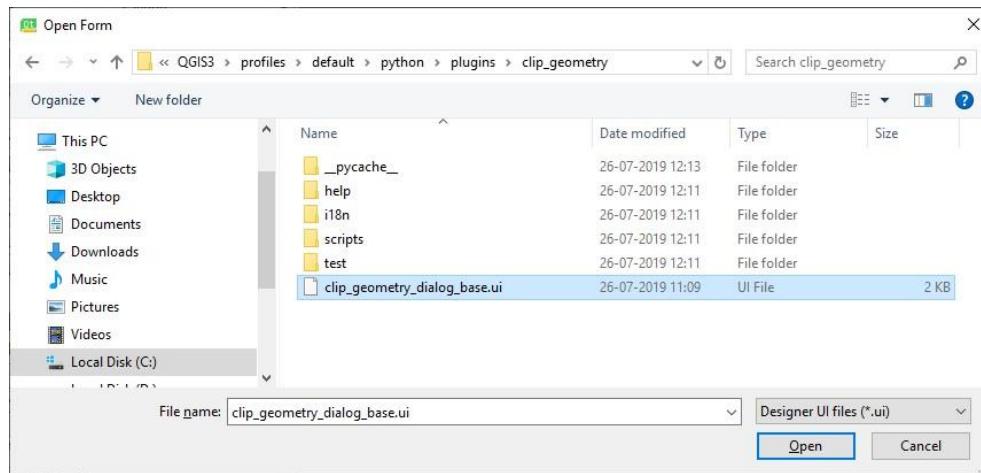


9. You will notice a new blank dialog named *Clip With Geometry*. Close this dialog.

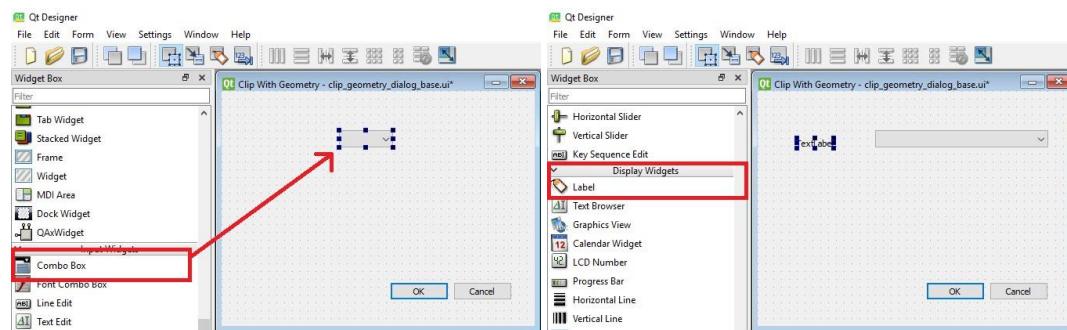


10. We will now design our dialog box and add some user interface elements to it. Open the Qt Designer program and to to *File -> Open File or Project....*

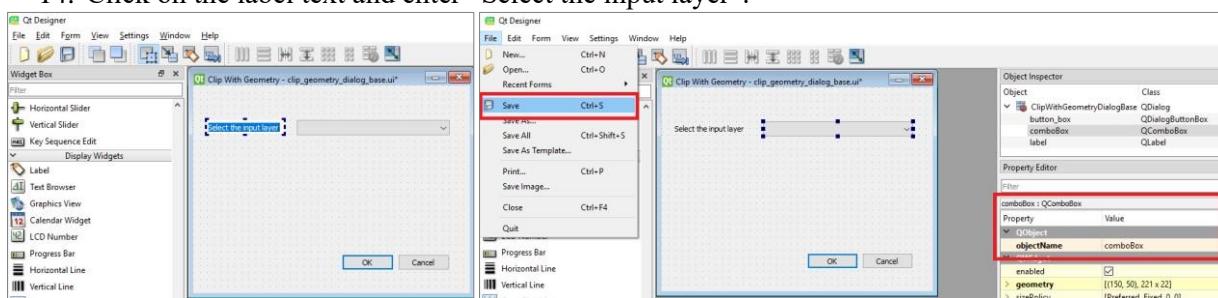
11. Browse to the plugin directory (in my case it is “C:\Users\prasun\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\clip\_geometry”) and select the clip\_geometry\_dialog\_base.ui file. Click *Open*.



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.

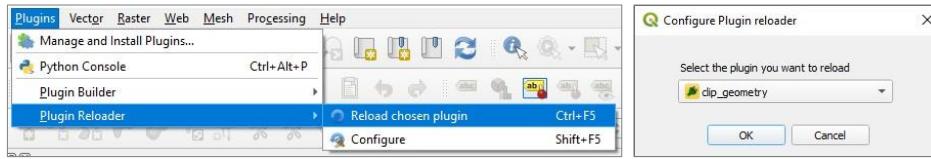


13. Resize the combo box and adjust its size. Now drag a *Label* type *Display Widget* on the dialog.
14. Click on the label text and enter “Select the input layer”.

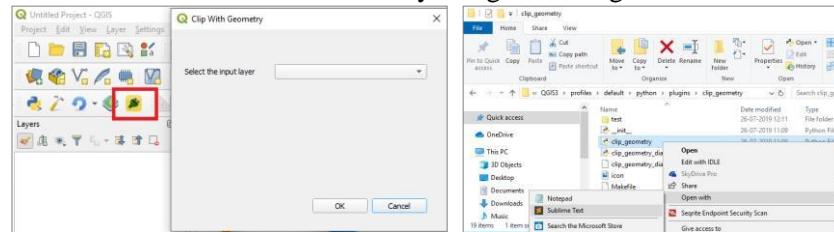


15. Save this file by going to *File - Save* “clip\_geometry\_dialog\_base.ui”. Note the name of the combo box object is comboBox. To interact with this object using python code, we will have to refer to it by this name.

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 ClipWithGeometry in the *Configure Plugin reloader* dialog.



18. Now click the button. You will see the newly designed dialog box.



19. Let's add some logic to the plugin that will populate the comboBox with the layers loaded in QGIS. Go to the plugin directory and load the file `clip_geometry.py` in a text editor. First, insert at the top of the file with the other imports:

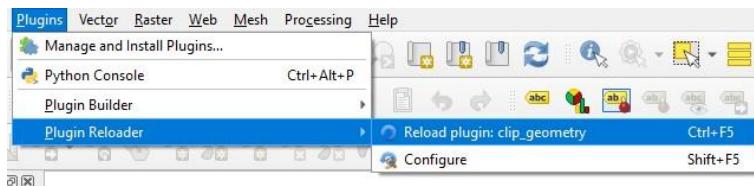
```
1.9 *   the Free Software Foundation; either version 2 of the License, or
2.0 *   (at your option) any later version.
2.1 *
2.2 ****
2.3 """
2.4 from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication
2.5 from qgis.PyQt.QtGui import QIcon
2.6 from qgis.PyQt.QtWidgets import QAction
2.7 from qgis.core import QgsProject
2.8
2.9 # Initialize Qt resources from file resources.py
3.0 from .resources import *
3.1 
```

20. 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. Save the file.

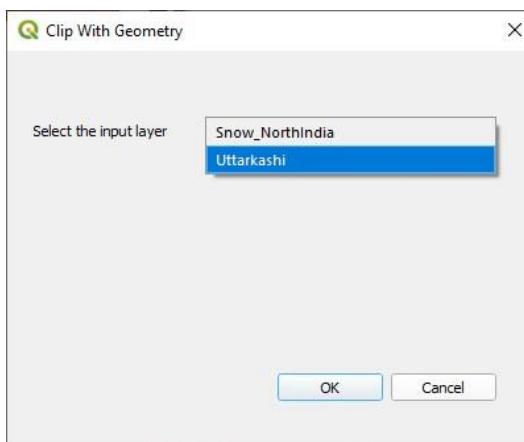
```
# Fetch the currently loaded layers
layers = QgsProject.instance().layerTreeRoot().children()
# Clear the contents of the comboBox from previous runs
self.dlg.comboBox.clear()
# Populate the comboBox with names of all the loaded layers
self.dlg.comboBox.addItems([layer.name() for layer in layers])
```

```
181     self.iface.removeToolBarIcon(action)
182
183
184     def run(self):
185         """Run method that performs all the real work"""
186
187         # Create the dialog with elements (after translation) and keep reference
188         # Only create GUI ONCE in callback, so that it will only load when the plugin is started
189         if self.first_start == True:
190             self.first_start = False
191             self.dlg = ClipWithGeometryDialog()
192
193         # Fetch the currently loaded layers
194         layers = QgsProject.instance().layerTreeRoot().children()
195         # Clear the contents of the comboBox from previous runs
196         self.dlg.comboBox.clear()
197         # Populate the comboBox with names of all the loaded layers
198         self.dlg.comboBox.addItems([layer.name() for layer in layers])
199
200         # show the dialog
201         self.dlg.show()
202         # Run the dialog event loop
203         result = self.dlg.exec_()
204         # See if OK was pressed
205         if result:
```

21. Back in the main QGIS window, reload the plugin by going to *Plugins - Plugin Reloader - Reload plugin: ClipWithGeometry*. Alternatively, you can just press Ctrl+F5.



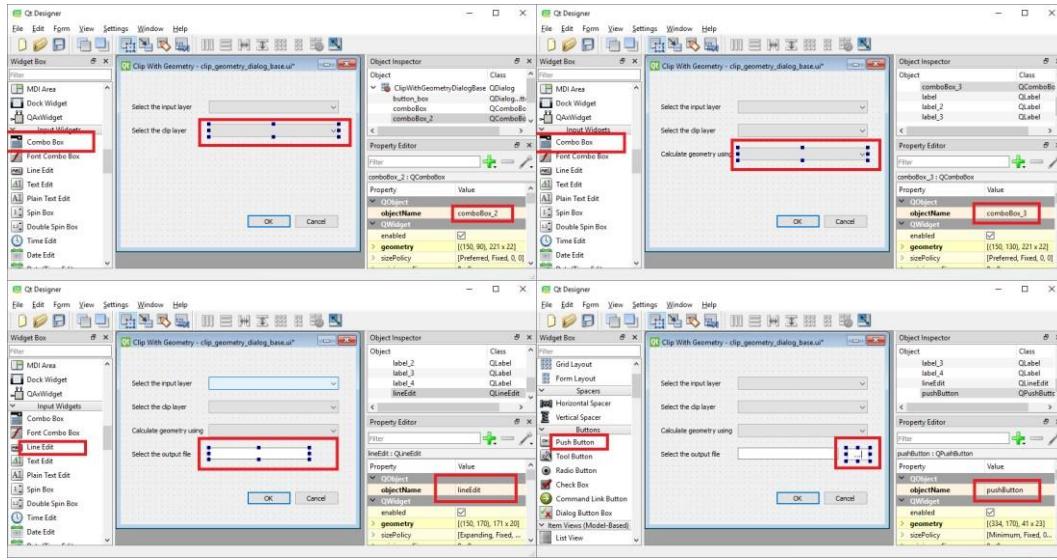
21. To test this new functionality, we must load some layers in QGIS. Add the two layers provided (Snow\_NorthIndia and Uttarkashi).
21. After you load the data, launch the plugin by going to *Vector - Clip With Geometry - Clip with Updated Geometry*. You will see that our combo box is now populated 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 *clip\_geometry.ui* file.

- Add a Label *Display Widget* and change the text to “Select the clip layer”.
- Add another Combo Box type *Input Widget* that will show the clip layer (objectName = *comboBox\_2*).
- Add a Label *Display Widget* and change the text to “Calculate Geometry using”.
- Add another Combo Box type *Input Widget* that will show the method used to calculate the Geometry values (objectName = *comboBox\_3*).
- Add a Label Display Widget and change the text to “Select the output file”.
- Add a LineEdit type *Input Widget* that will show the output file path that the user has chosen (objectName = *lineEdit*).
- 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 (objectName = *pushButton*). Save the file.

**THE OBJECT NAMES MUST BE AS SHOWN.**



19. Open the `clip_geometry.py` file in a text editor. Scroll down and find the `run(self)` method. Add the following code lines as shown. The 1<sup>st</sup> line gets the layers loaded in QGIS and adds it to the `comboBox_2` object from the plugin dialog; and the 2<sup>nd</sup> line indicates the options available to the user to Calculate Geometry value. **Careful of the indentation!**

```
self.dlg.comboBox_2.addItems([layer.name() for layer in layers])
self.dlg.comboBox_3.addItems(["Layer CRS","Project CRS","Ellipsoid"])
```

```
# Fetch the currently loaded layers
layers = QgsProject.instance().layerTreeRoot().children()
# Clear the contents of the comboBox from previous runs
self.dlg.comboBox.clear()
# Populate the comboBox with names of all the loaded layers
self.dlg.comboBox.addItems([layer.name() for layer in layers])
self.dlg.comboBox_2.addItems([layer.name() for layer in layers])
self.dlg.comboBox_3.addItems(["Layer CRS","Project CRS","Ellipsoid"])
# show the dialog
```

20. 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. Add `QFileDialog`, `processing` and `tempfile` to our list of imports at the top of the `clip_geometry.py` file.

```
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication
from qgis.PyQt.QtGui import QIcon
from qgis.PyQt.QtWidgets import QAction, QFileDialog
from qgis.core import QgsProject
import processing, tempfile
```

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, _filter = QFileDialog.getSaveFileName(self.dlg, "Select output file","", "*.shp")
    self.dlg.lineEdit.setText(filename)

```

```

    # Create the dialog with elements (after translation) and keep reference
    # Only create GUI ONCE in callback, so that it will only load when needed
    if self.first_start == True:
        self.first_start = False
        self.dlg = ClipWithGeometryDialog()
        self.dlg.pushButton.clicked.connect(self.select_output_file)

    # Fetch the currently loaded layers
    layers = QgsProject.instance().layerTreeRoot().children()

```

25. Now we need to add code so that when the ... button is clicked, `select_output_file` method is called. Scroll down to the `run` method and add the following line in the block where the dialog is initialized. This code will connect the `select_output_file` method to the clicked signal of the push button widget.

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

```

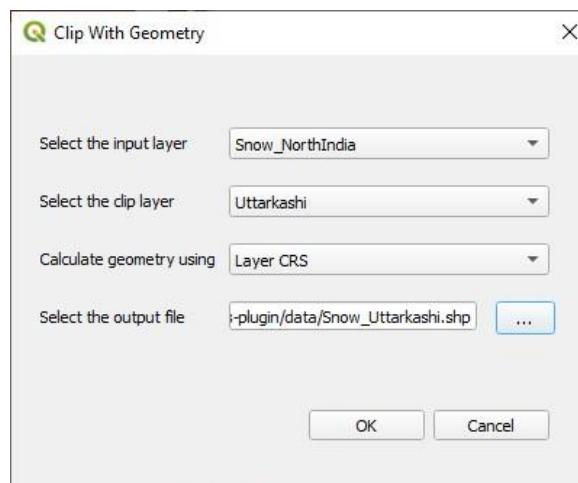
def run(self):
    """Run method that performs all the real work"""

    # Create the dialog with elements (after translation) and keep reference
    # Only create GUI ONCE in callback, so that it will only load when needed
    if self.first_start == True:
        self.first_start = False
        self.dlg = ClipWithGeometryDialog()
        self.dlg.pushButton.clicked.connect(self.select_output_file)

    # Fetch the currently loaded layers
    layers = QgsProject.instance().layerTreeRoot().children()

```

26. Back in QGIS, reload the plugin and open the *Clip With Geometry* 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.

```
infile = str(self.dlg.comboBox.currentText())
clipfile = str(self.dlg.comboBox_2.currentText())
geometry_method = self.dlg.comboBox_3.currentIndex()
outputfile = self.dlg.lineEdit.text()
intermediatefile=tempfile.NamedTemporaryFile(suffix='.shp')).name
processing.run("native:clip", { 'INPUT':infile, \
    'OVERLAY':clipfile, \
    'OUTPUT':intermediatefile})
processing.run("qgis:exportaddgeometrycolumns", { 'INPUT':intermediatefile, \
    'CALC_METHOD':geometry_method, \
    'OUTPUT':outputfile})
```

```
if result:
    # Do something useful here - delete the line containing pass and
    # substitute with your code.
    infile = str(self.dlg.comboBox.currentText())
    clipfile = str(self.dlg.comboBox_2.currentText())
    geometry_method = self.dlg.comboBox_3.currentIndex()
    outputfile = self.dlg.lineEdit.text()
    intermediatefile = (tempfile.NamedTemporaryFile(suffix='.shp')).name
    processing.run("native:clip", { 'INPUT':infile, \
        'OVERLAY':clipfile, \
        'OUTPUT':intermediatefile})
    processing.run("qgis:exportaddgeometrycolumns", { 'INPUT':intermediatefile, \
        'CALC_METHOD':geometry_method, \
        'OUTPUT':outputfile})
```

28. Now our plugin is ready. Reload the plugin and try it out. You will find that the output shape file will have the updated geometry columns after the clip operation.

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<sup>2</sup> so that all QGIS users will be able to find and download your plugin.

#### Clipping before Geometry Update

	DN	POLY_ID	AREA	PERIMETER
0	1	79	1062920248.453...	202211.936380
1	1	80	2943422164.273...	418569.344618

#### Clipping after Geometry Update

	DN	POLY_ID	AREA	PERIMETER
0		79	691890806.229675	173261.344021
1		80	967888489.997009	235742.687578

<sup>2</sup> <https://plugins.qgis.org/>

**Additional Problem:**

Create two overlapping polygon vector shapefiles in WGS-84 “geographic” coordinate system. Use the newly created tool to clip the layers. Change the options “Calculate Geometry using” and see the difference in the output attribute table. Interpret the results.

Material developed by Mr. Prasun Kumar Gupta, Geoinformatics Department, IIRS/ISRO

[prasun@iirs.gov.in](mailto:prasun@iirs.gov.in)