



# PyQGIS Cloud Masking Assistance Tool

*Development report of a cloud masking assistance tool written in Python for QGIS environment*

2018



**van hall  
larenstein**  
university of applied sciences



# PyQGIS Cloud Masking Assistance Tool

*Development report of a cloud masking assistance tool written in Python for QGIS environment*

<b>Type document:</b>	Method of approach
<b>Company:</b>	eLEAF
<b>Educational institution:</b>	University of Applied Sciences Van Hall Larenstein (VHL)
<b>Faculty:</b>	Land- and Water management
<b>Specializations:</b>	Applied Hydrology and Geoinformatics
<b>Place</b>	Hesselink van Suchtelenweg 6, Wageningen
<b>Date published:</b>	29 January 2018
<b>Author:</b>	William Tjong <a href="mailto:willy.tjong@hvhl.nl">willy.tjong@hvhl.nl</a>
<b>Supervision eLEAF:</b>	Kristin Abraham <a href="mailto:kristin.abraham@eleaf.com">kristin.abraham@eleaf.com</a>
<b>Supervision University (VHL):</b>	Jack Schoenmakers <a href="mailto:jack.schoenmakers@hvhl.nl">jack.schoenmakers@hvhl.nl</a>



**van hall  
larenstein**  
university of applied sciences



# Content

1	Introduction.....	1
1.1	Background.....	1
1.2	Problem statement.....	1
1.3	Objective.....	1
1.4	Outline of this document .....	2
1.5	Target audience.....	2
2	Methodology .....	3
2.1	Technical design .....	3
2.2	Key functions development.....	3
2.3	Satellite imageries dataset .....	4
2.4	GUI development .....	5
3	Core development.....	7
3.1	Python modules invoked in key functions .....	7
3.2	Key functions concept .....	8
4	Plugin development .....	15
4.1	Plugin structure .....	15
4.2	Building the plugin.....	16
5	Results .....	20
5.1	UI design.....	20
5.2	Basic workflow.....	21
6	Discussion .....	23
6.1	Bugs .....	23
6.2	Future use.....	23
	References.....	25
	Appendix 1 Screen dumps during development .....	26

# 1 Introduction

## 1.1 Background

Remote Sensing is the science of obtaining and analysing information about objects or areas from a distance (NOAA, 2017). Satellite-based remote sensing is applied in earth observation to gather information about the earth's physical, chemical and biological properties. These properties are able to emit or reflect electromagnetic radiation (EMR), e.g. infrared, into the space. Remote sensors on the satellite are able to identify this information. Since the launch of the first satellite in 1957, there are dozens of satellites orbiting and collecting data from the earth nowadays.

Remote sensing has become more important nowadays due to the potential to detect and prevent natural hazards. Besides hazard assessment, remote sensing can also be used for natural resources management (NOAA, 2017). The latter is a field of study in which eLEAF, a Wageningen based remote sensing company, is working in.

eLEAF supplies near-real-time data to the agriculture and water sector on crop evaporation and biomass production. Their aim is to support sustainable water use, increase food security and protect environmental systems by delivering accurate and reliable data on crop and water conditions. The company extract and analyse data from satellite imageries by means of eLEAF's PiMapping® technology. With this technology, data regarding biomass production, crop water use, water deficit and weather conditions can be derived (eLeaf, 2017).

## 1.2 Problem statement

Satellite imageries may contain clouds which can disturb the analysis process. The presence of clouds causes interference in the reflectance and absorption of shortwave radiation (e.g. visible light and UV) and the absorption and re-emission of longwave radiation (e.g. infrared and radio waves) (Jedlovec, 2009). This affects the quality of data, therefore contamination of cloud pixels in satellite imageries must be filtered (extracted) prior to processing.

To extract the cloud pixels from the image. A mask of the clouds and the cloud's shadows must be created. This can be done automatically or manually. The first is faster but does not guarantee the removal of all clouded parts, moreover non-cloud-objects may be masked out as well which results into the loss of valuable data. The company uses manual masking, this process includes visually assessing and editing the clouds by hands in QGIS. This process is inefficient and the company wants to improve it to increase productivity.

## 1.3 Objective

The objective of this research is to develop a tool for QGIS. This tool must cope with the following main features:

- Keep tracks of what has been done and still needs to
- Easy navigation through image parts
- Flexible zoom ranges for objective editing and having overview
- Alert users when working with multiple projections

The development of this tool will be documented in this report and the tool itself will be built in Python in QGIS environment.

### 1.4 Outline of this document

This document is composed of six chapters. The methodology of the tool development will be discussed in chapter two. Afterward, the core and plugin developments will be summarized in chapters three and four. Subsequently, the final version of the tool will be reviewed in chapter five. Finally, a few notes regarding future use and bugs will be discussed in chapter six.

### 1.5 Target audience

This document is written for the company eLEAF, lecturers of the University of Applied Sciences Van Hall Larenstein (VHL) and fellow students.

## 2 Methodology

In this chapter, the methodology in developing a Cloud Masking Assistance Tool will be explained. The development will be done in QGIS environment and written in Python. Sample images were provided by eLEAF to test the tool.

### 2.1 Technical design

To achieve the main functionalities mentioned in paragraph 1.3 in QGIS, a technical design was made and shown below. The diagram shows both key functions and its features.

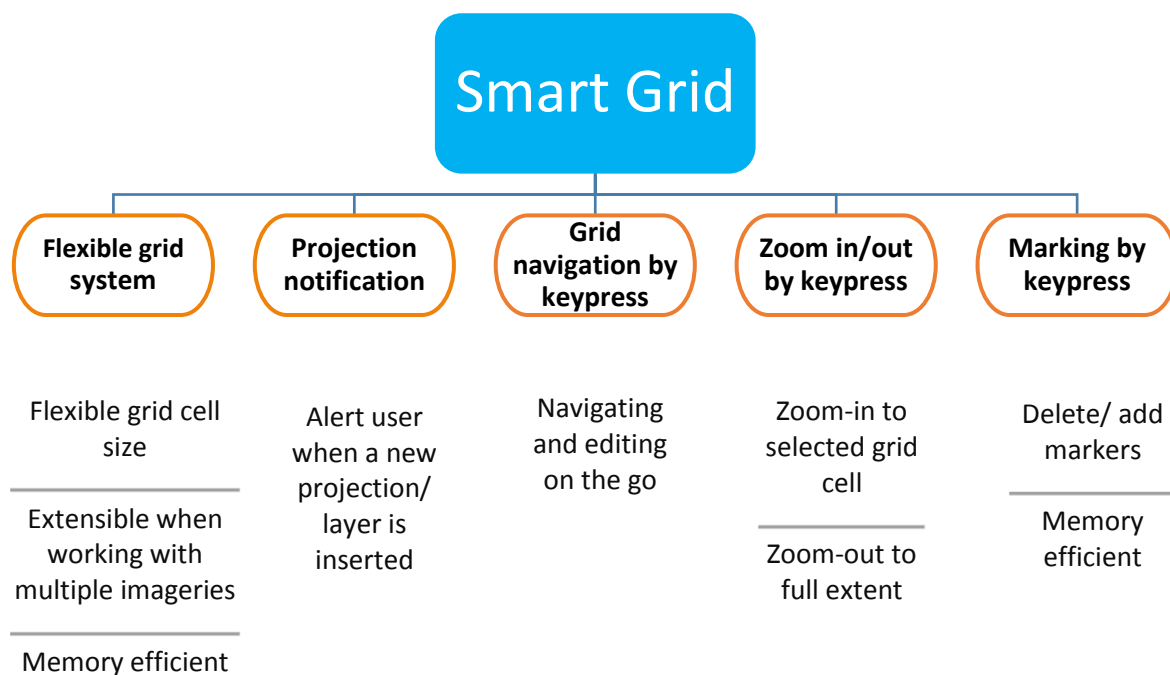


Figure 1 Technical Design of the Cloud Masking Assistance Tool also named as Smart Grid

### 2.2 Key functions development

QGIS is written in C++ and built with the use of the Qt Designer. Qt Designer is a cross-platform application framework that is used for developing application software (Qt, 2017). In QGIS the user can extend the key functionality of the application and write scripts to automate various tasks. The user can use either C++ or Python to write scripts in QGIS.

To build the key functions mentioned in paragraph 2.1, the Python programming language was used. Python is a high-level programming language that is well known for its readability and wide field of application. To interact with objects in QGIS interface the Python QGIS API was used extensively. PyQGIS API is built on top of the PyQt4 API. This API is used for graphical operations (i.e. navigation by keypress) but also for connecting functions and buttons on the GUI. Furthermore, standard Python modules for accessing the operating system and executing basic operations were used as well.

## QGIS Cloud Masking Assistance Tool

The key functions were developed for QGIS 2.18.15, but for backward compatibility, the 2.14.21 release was tested in some cases as well. Both releases come with Python 2.7.5 pre-installed.

### 2.3 Satellite imagery dataset

To test the cloud masking assistance tool, sample data from Sentinel2 (10m resolution), Landsat8 (30m resolution) and MODIS (250m resolution) satellite imagery were used. Screen dumps of these sample areas are shown below. Some of these data were already processed into NDVI maps. For editing purposes and improving rendering performance, lower resolution copies (8 times smaller resolution) of the images were build.

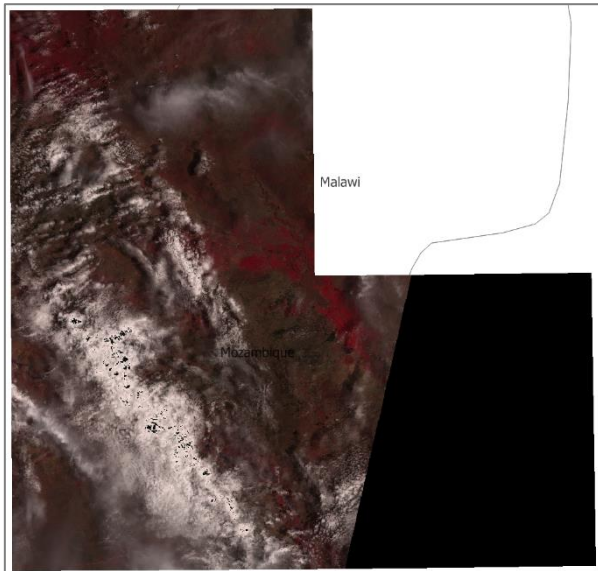


Figure 2 Three Sentinel2 images (15-11-2017) of Illovo Sugar Estate in Nchalo to the south of Malawi. CRS: WGS 84/ UTM Zone 36S. Total scaled data size: 587 MB. (images have been scaled/resampled from 3300 MB total data size to 587 MB total data size)

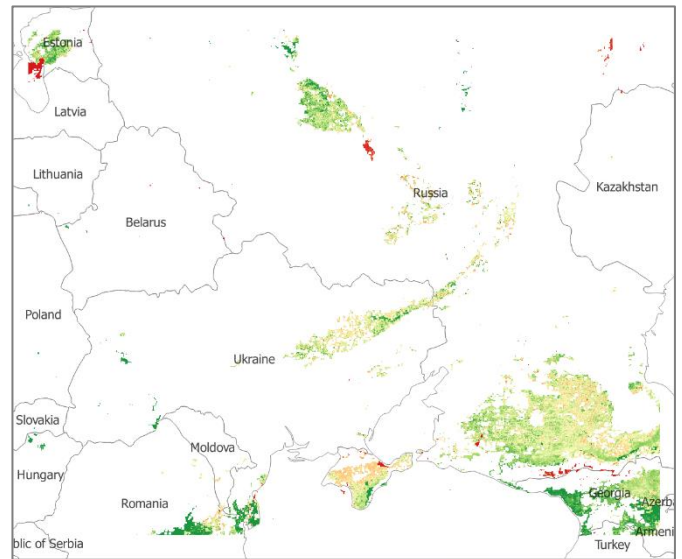


Figure 3 Two overlapping MODIS images (26-10-2017 & 12-11-2017) processed to NDVI of some areas in East Europe. CRS: custom (overlaps multiple UTM zones). Total data size: 594 MB.

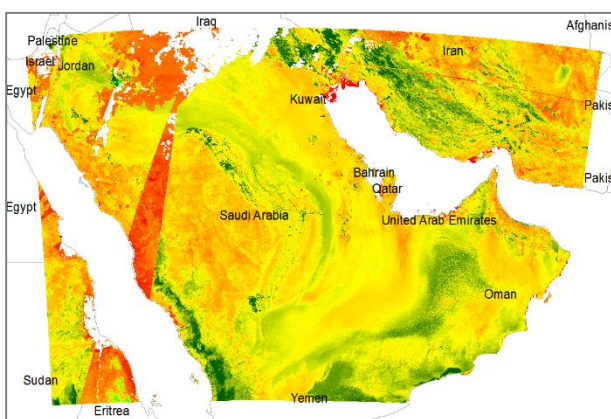


Figure 3 Two overlapping MODIS images (06-11-2017 & 08-11-2017) processed to NDVI of Saudi Arabia and surrounding countries. CRS: WGS 84/ UTM Zone 38N. Total data size: 1070 MB.

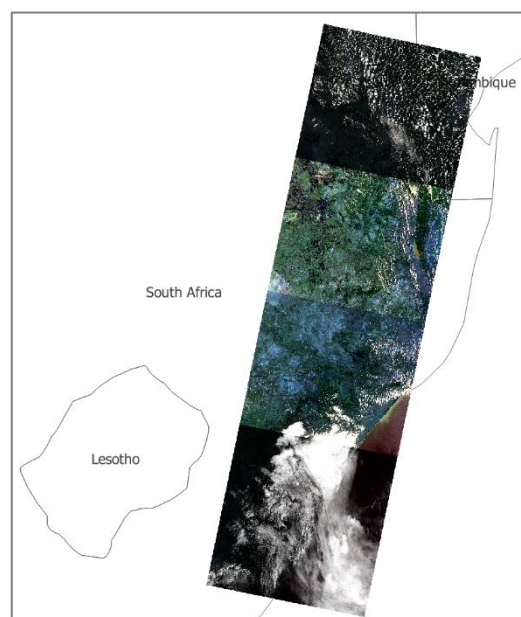


Figure 4 Four Landsat8 images (05-01-2015) of the east coast area of South Africa. CRS: WGS 84/ UTM Zone 36N. Total data size: 4340 MB.

### 2.4 GUI development

To create interactivity between the user and the key functions, an elementary interface by means of using Qt Designer. To implement the key functions into a QGIS plugin, there are a few dependencies needed. These dependencies are shown below.

#### **Plugin Builder and Plugin Reloader Plugins**

Plugin Builder (version 2.16.0) and Plugin Reloader Plugins were used to build the tool conveniently. The Plugin Builder creates a template with all the necessary files and resource codes for a plugin. The Plugin Reloader was used to reload the plugin in QGIS workspace while editing the plugin code. Hence, changes in the plugin can be viewed immediately without having to restart QGIS.

#### **Qt Designer with QGIS custom widgets**

Qt Designer comes pre-installed with the installation of QGIS Standalone application as well. The designer application is used to build the graphical user interface (GUI) of the plugin. The user interface contains the functionalities mentioned in paragraph 1.3. The Qt Designer application can read the user interface file (.ui file) created by the Plugin Builder.

#### **Pb\_tool (Python package)**

To test and deploy the plugin, resource files from the Plugin Builder must be compiled first. The most recommended way is by using the 'pb\_tool' (plugin builder tool) Python package. For Python 2.7.x release, pb\_tool-2.0.1. distribution was installed through OSGeo4W Shell<sup>1</sup> by using the 'pip install' method. The pb\_tool will compile the resource file (.qrc extension) to Python file. The resource file is an XML based file which contains data regarding icon, design and translations (Qt17).

#### **OSGeo4W**

OsGeo4W is a bundle of open source geospatial software packages such as GDAL/OGR and GRASS. It is included when installing QGIS. To execute the 'pb\_tool' compiler, the OsGeo4W Shell (command line interpreter) is used. This application is also included in the OSGeo4W package.

---

<sup>1</sup> OSGeo4W comes pre-installed with QGIS. The command shell enables the user to install and update OSGeo4W packages easily. OSGeo4W is a collection of open source GIS software, including QGIS.



## GUI concept

A concept of the user interface is shown below. A few features are noted to explain how the user can interact with the key functions. The key functions listed below operate in the background, therefore they are not displayed on the interface.

- Projection notification
- Grid navigation
- Zoom in/out
- Marking/ filling the grid

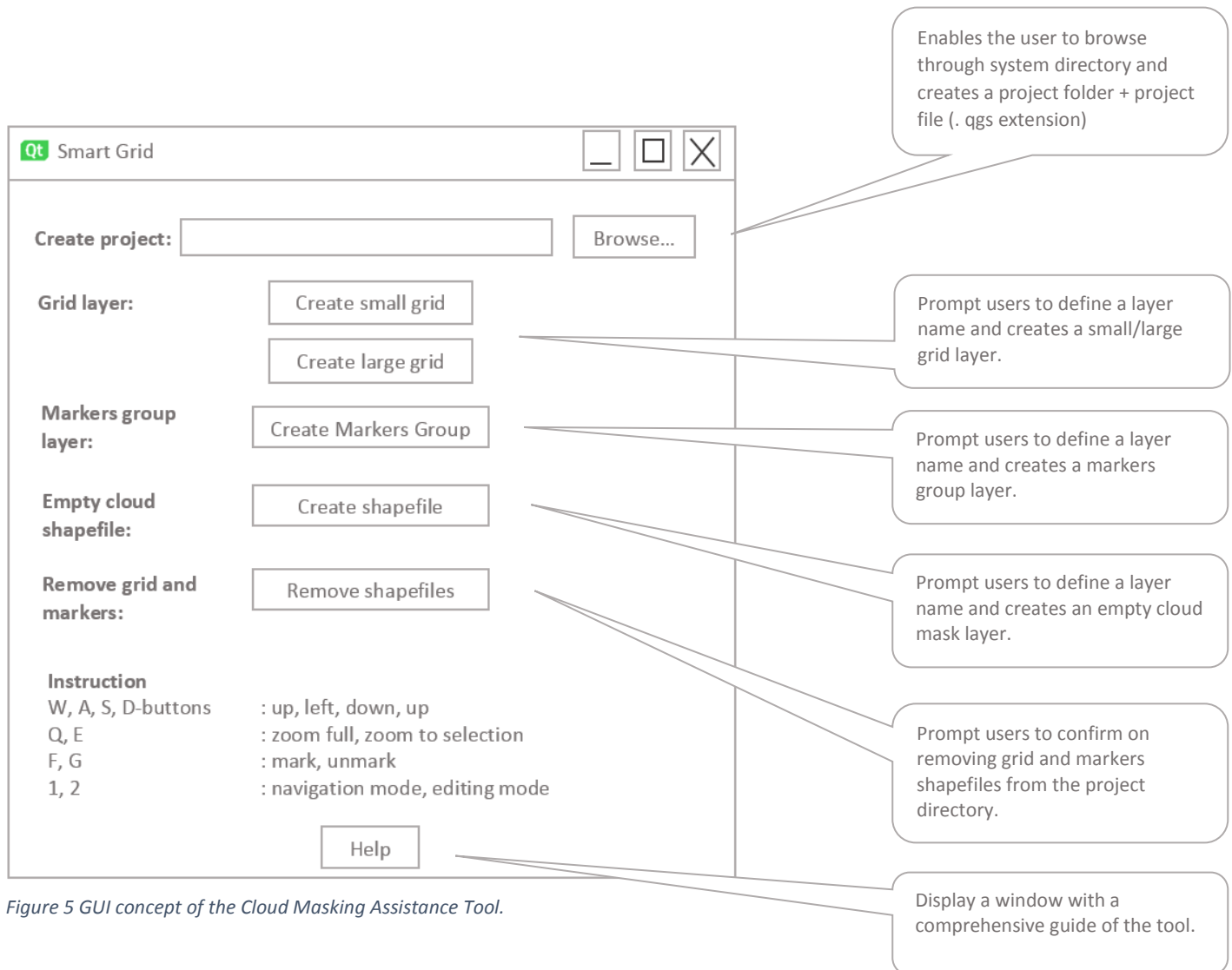


Figure 5 GUI concept of the Cloud Masking Assistance Tool.

### 3 Core development

Chapter three will provide an in-depth explanation about how the key functions work. The plugin structure will be discussed in this chapter as well.

#### 3.1 Python modules invoked in key functions

A few Python modules<sup>2</sup> were used to build the key functions. These modules are explained below.

##### 3.1.1 OS module

This module provides a method to access the operating system dependent functionality. It allows the user to interface with directories and underlying files. This module is used in the core functions to create a project folder and a QGIS project file (.qgs extension). It is also used to search and remove files in the project directory. This module comes pre-installed with QGIS 2.18.14 in Python 2.7.5 directory.

##### 3.1.2 PyQt4 package

PyQt is a popular Python binding which compiles Qt, C++ functionalities into Python format. There are two versions of PyQt: PyQt4 and PyQt5 (successor). QGIS 2.18.14 uses PyQt4 binding. The backbone of QGIS is built with PyQt4, which includes the QGIS GUI's widgets and buttons. There are a few modules and specific object/classes used in the key functions development in this package. These modules are QtGui and QtCore.

##### *QtGui module (PyQt4.QtGui)*

The QtGui module consists of a broad set of graphical user interface components. These classes are used in the key functions development:

- QAction: invoke a user interface action
- QColor: provides colour based on, among other things, the RGB colour model
- QShortcut: create keyboard shortcuts
- QKeySequence: encapsulates a key sequence as shortcuts

(PyQt Sourceforge, 2017)

##### *QtCore module (PyQt4.QtCore)*

The QtCore module provides non-GUI components used by other modules. For key functions development, the QFileInfo class is used to provide information about a file's name and path in a directory. The Qt class is used as a miscellaneous identifier for enumeration<sup>3</sup> or other classes in the Qt library. For example, 'Qt. Key\_3' enumeration is used to connect a function with a Windows Virtual-Key Code (symbolic constant names in hexadecimal values e.g. 0x01000000). (PyQt Sourceforge, 2017). The 'QVariant class' is also used for, among other things, to bond Qt data types (PyQt Sourceforge, 2017).

##### 3.1.3 QGIS package

Besides the PyQt4 package, the QGIS package provides an extension to the QGIS functionalities. The package enables user to manipulate spatial data and carry out analyses with it. From the QGIS package, the 'QgsMessageBar class' from the GUI (qgis.gui) module is invoked. This class is used to

---

<sup>2</sup> A module is a collection of variables, functions and classes (objects) which can be invoked in a Python script.

<sup>3</sup> A set of symbolic names bound to a constant/ unique value (Python Software Foundation, 2017).

communicate with the user (e.g. displaying pop-up dialogs). Furthermore, the 'Processing class' is also invoked to enable the user to use QGIS algorithms.

### 3.1.4 OSGeo package

The OSGeo package is a collection of open source software packages/ modules which provides extension to the QGIS functionalities as well. Most of these packages were written in C++ and compiled to Python format. The OGR (part of GDAL/OGR Python binding) module is used in the key functions development to manipulate vector data (OSGeo project, 2017).

## 3.2 Key functions concept

Multiple scripts and functions were developed which represent the key functions of the tool. The concept algorithm of these functions is discussed below.

### 3.2.1 Create project folder

This function creates a folder if a given folder path does not exist yet. The function also creates a QGIS project file every time the script is run. The algorithm concept of this function is summarized below.

```
#Folder creation
1. Open dialog and get selected directory location
2. Create a folder path with the selected directory location as
parent directory
3. If the folder path does not exist yet, create folder path
4. Else, do nothing

#Save project
1. Get active project
2. If folder path does not exist, save project in this path
3. Else, do nothing
```

### 3.2.2 Create small/ large grid layer

There are two methods to create a grid layer:

- Single layer grid → creates a vector grid for a single active raster layer
- Map extent grid → creates a vector grid for multiple raster layers

The single layer grid is created when a single raster layer is selected. The map extent grid is created when there are raster layers displayed within the bounding box of the map canvas. The grid layers are created using a QGIS algorithm ('qgis vectorgrid' algorithm). This algorithm requires the following arguments:

- Layer extent
- Grid cell size (x, y grid spacing)
- Output type (polygon or polyline)
- Output path (e.g. ~/Users/Downloads/Grid.shp)

The grid shapefile is saved in the project file directory. The grid layer contains extent parameters and an ID as attributes. The grid layer is saved in the project directory. The algorithm concept of the grid creation functions is shown on the next page.

## Single layer grid

```
#Vectorgrid creation
1.Get directory path of active project
2.Get active layer
3.If active layer is a raster layer, calculate arguments for the
'qgis vectorgrid' algorithm and execute it.
4.Else, do nothing

#Change symbology
1.If a layer in the layers panel is named 'Grid', change
borderline colour to blue
```

## Map extent grid

```
#Vectorgrid creation
1.Get directory path of active project
2.Create an empty list for layers visible in map canvas
3.Populate this list with only raster layers which are within the
extent of the canvas
4.Calculate qgis vectorgrid algorithm arguments for all the layers
in the list
5.Execute qgis vectorgrid algorithm

#Change symbology
1.If a layer in the layers panel is named 'Grid', change
borderline style colour to blue
```

### 3.2.3 Create grid marker layer

To keep tracks of work, the user can mark parts of an image. This is done by the created vectorgrid- and grid marker layers. The create grid marker layer is an empty polyline which can be populated with features. These features are created by keypress functions. The grid marker layer is saved in the project folder. The concept algorithm of the create grid marker function is explained below.

```
#Grid marker layer creation
1.Get directory path of active project
2.Get active layer
3.If directory path of active project exists and active layer is a
raster layer, create empty polyline layer.
4. Else, do nothing

#Change symbology
1.If a layer in the layers panel is named 'Grid marking', change
borderline style colour to green
```

### 3.2.4 Create cloud mask layer

This function creates an empty polygon layer and save it in the project directory as well. The layer is used as a cloud mask base layer. The concept algorithm of this function is summarized below.

```
#Polygon layer creation
1.Get directory path of active project
2.Get active layer
3.If active layer is a raster layer, create empty polygon layer
4.Else, do nothing
```

### 3.2.5 Create markers group layer

This function creates a group layer to group the grid marker and the vectorgrid. The markers group layer is not a vector but a QGIS layer tree object. Therefore, no shapefiles are created and saved in the project directory. The concept algorithm of this function is show below.

```
#Group layer creation
1.Get layer tree of active project
2.If group layer does not exist, create group layer
3.Else, delete previous group layer and create a new one

#Insert layers to group
1.If grid marker or grid layers exists, insert to group
2.Else, do nothing
```

### 3.2.6 Keypress functions

To edit a cloud mask more conveniently, the user can navigate through image parts. This can be done with the use keyboard shortcuts. A few keypress functions were developed to connect a keypress signal with a certain action. In addition, an autonomously function was developed as well to notify the user when working with multiple projections. These functions are explained in this paragraph.

#### Toggle between navigation end editing

By means of keypresses the user can toggle between navigation and cloud mask editing modes. Navigating can only be done on the vectorgrid layer. Editing is done on the cloud mask layer. The concept algorithm of these modes is explained below.

```
#Toggle to grid navigation mode
1.If there is a grid layer in the layers panel, set this layer as active
2.Close feature editing mode of the cloud mask layer and save edited features

#Toggle to cloud mask editing mode
1.If there is a cloud mask layer in the layers panel, set this layer as active
2.Activate feature editing mode of the cloud mask layer
```

The following keyboard keys are used:

- Number '1' key → Editing mode (activate cloud mask layer)
- Number '3' key → Navigation mode (activate grid layer)

## Zoom in/ out and deselect feature functions

The zoom in/out and deselect feature functions can be used on the vectorgrid layer only as well. The zoom out function, zooms out to the grid layer extent. The zoom in function, zooms in to a selected grid cell. The deselect feature function undo this selection. The concept algorithm of these functions is summarized below.

```
#Zoom out to layer extent
1.Get active layer (vector grid)
2.Get layer extent
3.Set layer extent as new extent

#Zoom in to selected feature
1.Get active layer (vector grid)
2.Zoom to selected grid cell feature

#Deselect feature
1.If there is a grid layer in the layers panel, remove any feature
selection on that layer
2.Else, do nothing
```

The following keyboard keys are used:

- Q key → Zoom to Layer
- E key → Zoom to Selection
- R key → Deselect Feature

## Mark/ unmark grid cell functions

The mark grid cell function adds a new feature to the grid marker layer by a keypress. The user can unmark/ delete this feature by a keypress as well. Deleting and adding features to the grid marker layer is done using the ID and extent attribute fields from the vectorgrid layer. Therefore, this function is non-destructive to the vectorgrid layer. It is editing the grid marker layer only. The concept algorithm of these functions is explained below.

```
#Mark a grid cell
1.Get active layer (vector grid)
2.If active layer is the grid layer, get ID and extent attributes
from selected grid cell
3.Create a rectangle geometry (feature) based on the extent
attributes of the selected grid cell
4.Assign an ID to this feature (ID from selected grid cell)
5.Save feature to the grid marker layer

#Unmark a grid cell
1.Get active layer (vector grid)
2.If active layer is the grid layer and the selected grid cell is
the same as the one on the grid marker layer, delete this feature
from the grid marker layer.
3.Else, do nothing
```

The following keyboard keys are used:

- F key → Mark
- G key → Unmark

## Fill grid cell(s) function

In certain cases, a large part of the satellite image may be clouded. The user must mask this part as well. The tool enables the user to fill a clouded grid cell instantaneously with a keypress. The fill grid cell function is non-destructive to the grid layer, because it is only editing the cloud mask layer. The concept algorithm of this concept is summarized below.

```
#Fill grid cell(s)
1.Create a list for select grid cell features
2.Populate this list with the ID- and extent attributes of the
selected grid cell.
3.Create a rectangle geometry (feature) for each selected feature
based on the extent attributes of this.
4.Assign an ID to each feature
5.Save feature to the cloud mask layer
```

The following keyboard keys are used:

- Number '2' key → Fill grid cell(s)
- G key → Delete fill

## Grid navigation function

An example of a vector grid is shown in figure 6 on the next page. The grid layer has an ID attribute field, which the navigation operation is based on. The index of the ID attribute field starts at the top left of the grid structure. Navigating through the grid cells requires the user to select a grid cell (select feature) first. If the user wants to navigate from left to right, it will either add or subtract '1' index from the selected grid cell. If the user wants to navigate upward or downward, it will either add or subtract the number of columns as index from the selected grid cell. Figure 6 on the next page displays a 9 x 9 (rows x columns) grid. Each time the user descends to the next row, all the indices in the current row is added with 9 which equals the indices in the next row. The navigation system can be explained in a formula as shown below.

- Right grid cell index = selected grid cell ID index + 1
- Left grid cell index = selected grid cell ID index - 1
- Above grid cell index = selected grid cell ID index - numbers of column
- Below grid cell index = selected grid cell ID index + numbers of column

The concept algorithm of this function is summarized on the next page.

The key buttons to navigate are listed below.

- W key → upward,
- A key → left
- S key → downward
- D key → right

```
#Move left/ right
1.Get active layer
2.If active layer is the grid layer, get ID of selected grid cell
3.Calculate right/left grid cell index
4.Set a new feature selection action based on the right/ left
calculated grid cell index

#Move upward/ downward
1.Get active layer
2.If active layer is the grid layer, calculate total number of
columns
3.Calculate above/below grid cell index
4.Set a new feature selection action based on the above/below
calculated grid cell index
```

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

Figure 6 Grid structure with the ID as labels.



### CRS notification function (autonomous function)

The CRS notification function operates autonomously. This function search for difference in projections between all raster layers in the layers panel when added. A message bar displays the projections from the raster layers which are available in the layers panel. The concept algorithm of this function is summarized below.

```
#CRS alert
1.Create a list to store layers
2.Populate list with raster layers
3.Create a list for to store the projection reference system of
the raster layers
4.Removes projection duplicates from this list
5.If there are multiple projections in this list, show a message
bar
6.Else, do nothing
```

### 3.2.7 Remove files

This script removes the grid- and grid marking shapefiles from the project folder. It does not delete the QGIS project file and the cloud mask shapefiles. The concept algorithm of this function is summarized below.

```
#Remove files
1.Get directory path of active project
2.If directory path exists, delete 'Grid' and 'Grid_marking'
shapefiles
3.Else, do nothing
```

## 4 Plugin development

The plugin development will be discussed in this chapter. This includes the creation of the plugin files with the Plugin Builder plugin, compilation of these files. Subsequently, the implementation of the key functions, explained in chapter 3, will be reviewed as well.

### 4.1 Plugin structure

A PyQGIS plugin is a type of Python package which has certain modules and functions. The difference between a regular Python package and PyQGIS plugin package is that it has an interface and the codes within this package only works for that plugin. To create a PyQGIS plugin it is recommended to use the Plugin Builder plugin. This plugin creates a plugin template in the QGIS plugin directory. The template consists of a few source files to execute the plugin in QGIS environment. A few of these files are summarized below.

#### 4.1.1 `__init__.py`

Every Python package has an '`__init__.py`' file. This file is required to treat the directory as a Python package. Besides, it is also used to execute initialization codes<sup>4</sup>. When developing a standalone Python package this file can be left empty. However, in developing a PyQGIS plugin, the file must contain at least a '`classFactory ()`' method/ class object. This method basically imports a function from the `mainPlugin.py` and convert the function into an 'interface' object. This object can be connected to, for example a push button on the interface (QGIS , 2017).

#### 4.1.2 `mainPlugin.py`

The main operational file of the plugin. It contains all the code about the actions of the plugin and the main functions (QGIS , 2017). File name is customizable. This file contains the plugin class object with the following attributes/ methods:

```
__init__ (self, iface)
```

This attribute is a 'constructor'. A constructor is used to initialize a newly created object. Function within this attribute will be invoked upon creation of a new object. The first argument is always 'self', which is a reference to the object being initialized. Secondary argument in this plugin is 'iface' which is a reference to the interface. This interface allows access to the map canvas, menus and other parts on the QGIS application (Vallecha, 2015).

```
tr (self, message)
```

This attribute is used to translate string objects in the plugin. It uses the Qt translation API.

```
add_action (self, *)
```

Multiple arguments\* are required within this attribute:

- icon\_path
- text
- callback
- enabled\_flag= True
- add\_to\_menu=True
- add\_to\_toolbar=True
- status\_tip= None

---

<sup>4</sup> Initializing in Python means assigning values to any instance variables that an object will need when it starts. (Dummies, 2017)

- `whats_this=None`
- `Parent=None`

This attribute can be used to specify a special state of the element on the plugin interface. For example, a status tip pop-up when the user hovers its mouse pointer over a push button.

`initGui (self)`

This attribute creates the menu entries and toolbar icons inside the QGIS GUI.

`onClosePlugin (self)`

This attribute clean-ups (un)necessary items when the plugin dialog/ dock widget is closed. For example, removing in memory layers or variables when the plugin dialog/ dock widget is closed or terminating certain processes.

`unload (self)`

This attribute removes the plugin menu item and icon from QGIS GUI.

`run (self)`

This attribute load and starts the plugin.

#### 4.1.3 resources.qrc and resources.py

The 'resources.qrc' is a .xml type document created by the Qt Designer. It contains relative paths to resources (e.g. plugin images, logos or icons) of the 'form.ui' file. The 'resources.py' file is the translation (compiled file) of the 'resources.qrc' file in Python (QGIS , 2017).

#### 4.1.4 form.ui and form.py

The 'form.ui' contains the GUI objects created by the Qt Designer. The 'form.py' file is the compiled file of the 'form.ui' file in Python (QGIS , 2017). File name is customizable.

#### 4.1.5 metadata.txt

Contains general info, version, name and some other metadata used by plugins website and plugin infrastructure (QGIS , 2017).

### 4.2 Building the plugin

To create the source files summarized before, a few steps were carried out. These steps are summarized below.

1. Fill in the QGIS Plugin Builder form
2. Compile and deploy plugin with the `pb_tool` via OSGeo4W Shell
3. Edit the interface source file (.ui extension) with Qt Designer
4. Implement key functions in main operational Python file (e.g. `mainPlugin.py`)

#### 4.2.1 Plugin Builder form

A result dialog (see appendix 1.1) is shown after filling in the Plugin Builder form. This dialog shows, among other things, where the plugin folder is created and couple notes regarding development.

#### 4.2.2 Compile and deploy

A screenshot of the use of the 'pb\_tool' is shown in appendix 1.2. The 'pb\_tool deploy' command is executed via the OSGeo4W Shell. This action requires a configuration file named 'pb\_tool.cfg' from

the plugin's folder. This configuration file is created with the aid of the Plugin Builder plugin. The file is needed for the pb\_tool to look up which files are to be compiled.

A before and after compilation/ deployment folder structure is shown in appendix 1.3. Only the essential files, which were mentioned in paragraph 4.1, are left after the compilation.

#### 4.2.3 GUI design

A screenshot of the GUI development can be seen in appendix 1.4. Object names will be invoked in the smart\_grid.py script (main operational file). User interface has been customized using CSS in Qt Designer.

#### 4.2.4 Implementation of key functions

##### Additional Python modules

Key functions implementation is explicitly done in the smart\_grid.py file. Various qgis and PyQt4 modules are already set up before implementation. A before and after screen dumps of these modules are shown in figure 7 and 8 below. Modules used in the key functions were explained in paragraph 3.1, some of them are invoked in the main operational file as shown in figure 8. Other modules are invoked within the key function scripts.

```
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication, Qt
from PyQt4.QtGui import QAction, QIcon
# Initialize Qt resources from file resources.py
import resources_rc

# Import the code for the DockWidget
from smart_grid_dockwidget import SmartGridDockWidget
import os.path
```

Figure 7 Before key functions implementation

```
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication, Qt, QFile, QFileInfo
from PyQt4.QtGui import QAction, QIcon, QFileDialog, QMessageBox, QShortcut, QKeySequence, QColor
from PyQt4 import QtGui
from qgis.utils import *
from qgis.core import *
from qgis.gui import QgsMessageBar

#key function scripts
from create_large_grid import largeGrid
from create_group import markersGroup
from empty_cloud import createCloud
from remove_files import removeFiles
from crs_alert import alert

from mark_unmark2 import markCell, unmarkCell
from navigation import forward, backward, upward, downward
from nav_edit import grid_active, cloud_active
from filler import fillCell

import os
import processing

# Initialize Qt resources from file resources.py
import resources

# Import the code for the DockWidget
from smart_grid_dockwidget import SmartGridDockWidget
import os.path
```

Figure 8 Afterkey functions implementation

These scripts are called again in the main operational file (see comment line ‘#key function scripts’ in figure 8). The main plugin script connects each key function with a keypress or push button. A diagram of the scripts and its relationship with each other is shown in figure 9 below.

## Smart Grid Plugin

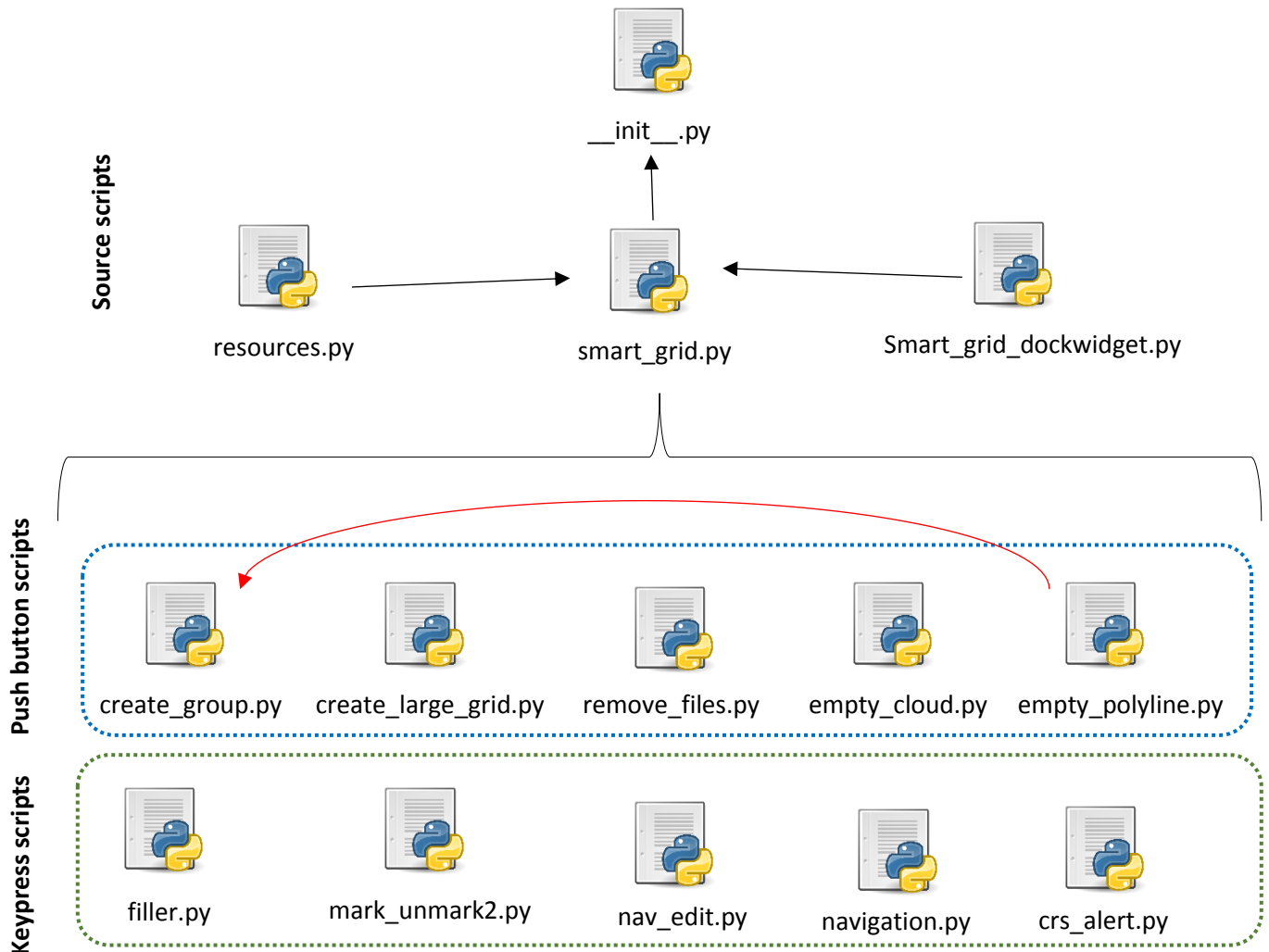


Figure 9 Smart Grid Plugin diagram showing the relationship between main operational file (`smart_grid.py`) and key function scripts. Key function scripts are grouped in Push button scripts (green) and Keypress scripts (blue). The `empty_polyline.py` is invoked in the `create_group.py` (red arrow).

Note, that there is the `create_large_grid.py` file in this diagram. This is a key function which creates a vector grid layer based on the raster layers within the map extent. The single raster layer vector grid is hardcoded in the `smart_grid.py` file, but this is part of the ‘Push button’ functions as well. Furthermore, these key functions are hardcoded in the `smart_grid.py` file as well:

- Create project folder (push button)
- Zoom to layer (keypress)
- Zoom to selected feature (keypress)
- Deselect feature (keypress)
- Grid cell size (combo box, additional)

### Main operational file and additional functions

As mentioned in paragraph 4.1.2 the main operational file consists of multiple attributes. During plugin development only the ‘\_\_init\_\_(self, iface)’ attribute was edited. In this attribute, push button objects and keyboard presses are connected to certain actions and functions. Examples of these operations are displayed below.

```
#dockwidget operations
self.dockwidget.large_grid.pressed.connect(largeGrid)
self.dockwidget.small_grid.pressed.connect(self.smallGrid)

#autonomous operation
QgsMapLayerRegistry.instance(). legendLayersAdded.connect(alert)

#keypress functions
#connect mark () function with the F button
shortcut = QShortcut (QKeySequence (Qt. Key_F),iface.mainWindow())
shortcut.setContext (Qt. ApplicationShortcut)
shortcut.activated.connect(markCell)
```

Dock widget operations refer to interface objects/ push buttons. Once pressed, a function is executed. Note that `(largeGrid)` is invoked from a script (see figure 9) and `(self.smallGrid)` is invoked from within the plugin class as an attribute.

The autonomous operation executes the ‘alert’ function (crs\_alert.py in figure 9) whenever a layer is inserted to the layers panel.

The keypress function connects a key press signal to a certain action as well. This action occurs on the QGIS interface.

## 5 Results

In this chapter, a few screen dumps of the plugin in action will be displayed. The plugin is named Smart Grid, because the key functions are based on a grid.

### 5.1 UI design

The user interface of the Smart Grid is a dockable window with three tabs. These tabs are displayed below.

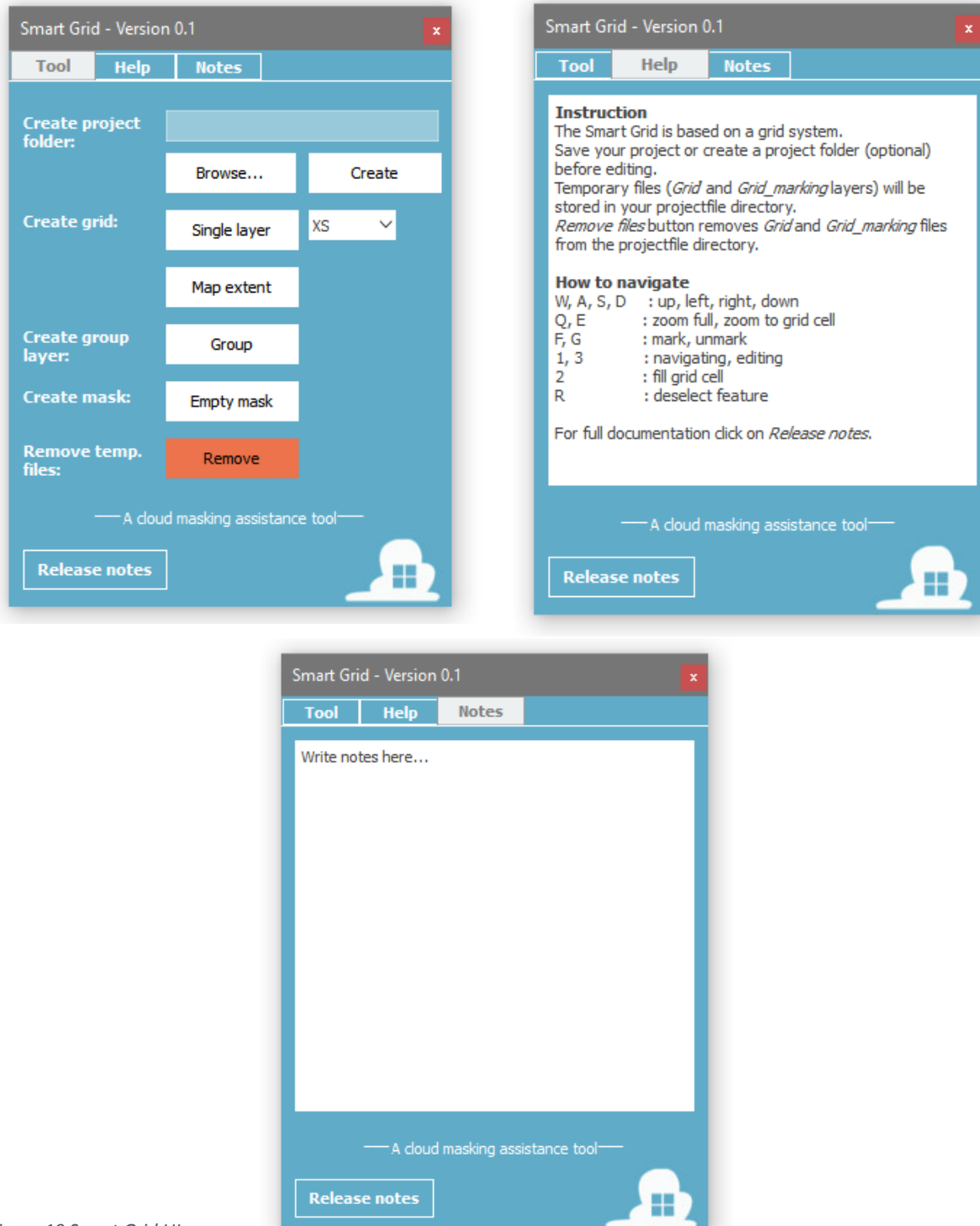


Figure 10 Smart Grid UI

### 5.1.1 Tool tab

The main tab consists of 8 push buttons, 1 combo box and 1 line edit objects. The push buttons connect a pressed button signal to a function or Python script. The browse button, enables the user to select a directory path. This path is presented in the line edit as a text/ string. The create button use this string to create a project folder named 'cloudmask'.

The single grid layer button creates a grid based on a selected aster layer. The combo box next to this button, enables the user to select the grid cell size of the grid layer.

The map extent grid button does not have an option to choose a grid cell size. This button creates a grid based on the visible raster layer(s) within the map canvas.

The group button, creates a group names 'Markers Group' and insert a "Grid\_marking" layer to it, which enables the user to mark the grid cells. It will prompt the user to define the projection system for the 'Grid\_marking' layer. If a grid layer is shown in the layers panel, this layer will be added to the group layer as well.

The empty mask button, creates a vector layer for the purpose of cloud masking. When pressed, the user is prompt to enter a layer name and a projection system. The layer name must contains the word/ string 'cloud'. For example, 'cloud\_9', 'cloud\_mask1', 'cloud1' and 'mask\_cloud' are correct layer names for this function.

The remove button removes 'Grid' and 'Grid\_marking' layers from the project path. The project path is the created project folder path or the path where the active project is saved.

### 5.1.2 Help and Notes tabs

The help tab contains a text frame with a brief instruction for the use of the Smart Grid tool.

The notes tab contains an editable text frame. In this tab the user can write notes if they like.

There is an additional release notes button which opens up a webpage to the documentation of the Smart Grid tool. This webpage will opens up in the default browser of the user (e.g. Chrome, IE9 or Safari). Note that this webpage is not a hosted page but is an html file in the plugin directory.

## 5.2 Basic workflow

The basic workflow for this tool is summarized below.

1. Browse through directory and select a directory path (Browse button)
2. Create a project folder (Create button)
3. Choose a grid cell size (Combo box)
4. Select a single raster layer from the layers panel
5. Create a single layer grid (Single layer button)
6. Create group and define the 'Grid\_marking layer's' projection system (Group button)
7. Select a single raster layer from the layers panel
8. Create an empty mask layer and define the 'cloud mask layer's' projection system (Empty mask button)



## 9. Navigating and editing keys:

- W,A,S,D : up, left, down, right a grid cell
- Q,E : zoom full, zoom to grid cell
- F, G : mark, unmark grid cell (delete cloud filled grid cell)
- 1, 3 : editing, navigating
- 2 : fill grid cell(s)
- R : deselect feature

10. When an image is done, remove Markers Group and cloud mask layer manually from the layers panel

11. Repeat

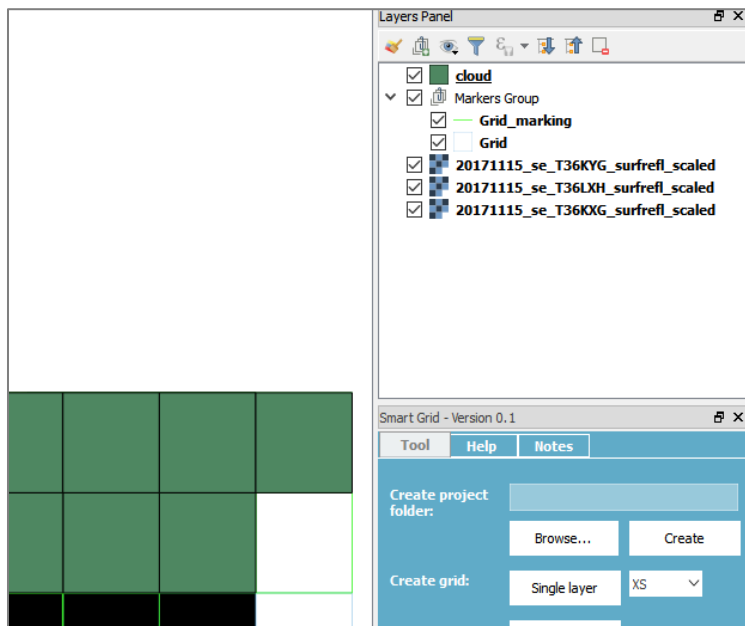


Figure 11 Basic workspace look.

If the user follows the basic workflow as mentioned before, their workspace should look like figure 11. The cloud mask layer should be on top of the layers panel. Afterward, the 'Markers Group' layer contains a 'Grid\_marking' layer on 1<sup>st</sup> position and a 'Grid' layer on 2<sup>nd</sup> position. Raster layers should be below all the above layers.

A status bar will appear on the bottom right corner of the QGIS interface when multiple layers with different projection systems are added to the layers panel. An example is shown in figure 12.

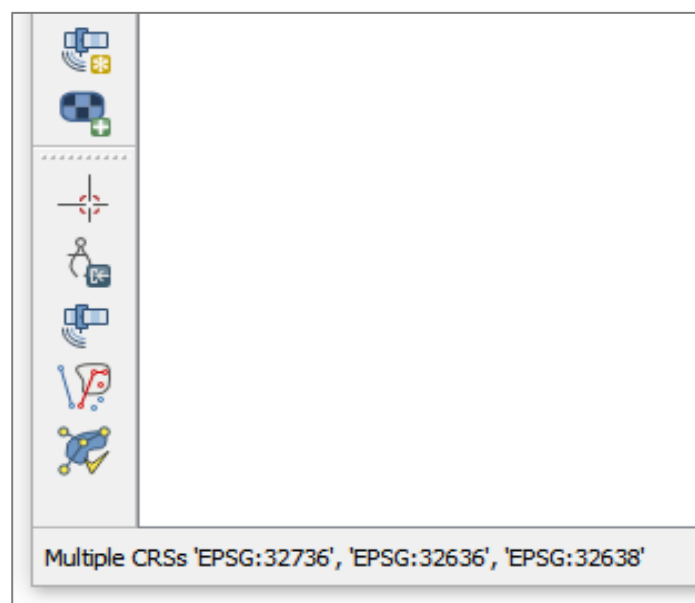


Figure 12 Basic workspace look.

## 6 Discussion

Several bugs in the Smart Grid tool will be summarized in this chapter. Beside, a brief review of what might change in the code in the future will be given. This is due to the upcoming release of QGIS version 3.0.

### 6.1 Bugs

Due to time constrain, it was not possible to fully develop the tool for optimal user experience. There are several bugs which should be improved for future use. These bugs are reviewed below.

#### 6.1.1 Termination of keypress functions

During runtime of QGIS, all plugins will be set up as well. From this moment, the navigation functions are 'activated'. Subsequently, the user is able to do graphical operations by keypresses (e.g. navigating on the grid layer).

The problem occurs when the user restart the Smart Grid plugin with the Plugin Reload plugin. This event will reload the plugin but also terminate the key press functions. However, the push buttons functions are still operational. The keypress functions are terminated as well when the keyboard key slot is occupied by a user-defined shortcut keys (Settings > Configure Shortcuts). As long as the user is not using the Plugin Reloader or the same shortcut keys as the Smart Grid tool, it should be fine.

#### 6.1.2 Unable to cloud fill a grid cell from a user newly created cloud mask layer

The number '2' keyboard key enables the user to fill a grid cell with. Basically, this function appends a new feature on the cloud mask layer. The fill grid cell function is able to append a feature based on the selected grid cell. This cloud fill feature also inherits/ copies the ID of the selected grid cell. This is needed to delete it again when the user presses the 'G' keyboard key.

The bug occurs when a user creates its own empty cloud mask layer with the 'New Shapefile Layer...' option from the QGIS Manage Layers Toolbar (see figure 13). The cloud fill feature will not be displayed on the map canvas. However, the ID is copied only.

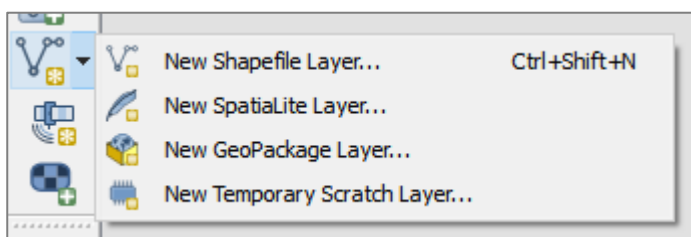


Figure 13 New Shape File Layer option from the QGIS Manage Layers toolbar

### 6.2 Future use

Development of QGIS happens periodically. According to the QGIS homepage, a new release happens every 4 months. These releases have even version numbers, e.g. 2.18, 3.2 etc. Every third release a LTR (stable release) is established. These LTRs are well maintained by the QGIS developers. The current LTR is QGIS version 2.14.22. There is also the regular latest release of QGIS version 2.18.16. (QGIS, 2017)

### 6.2.1 Backward compatibility QGIS 2.14.x and 2.18.x

The Smart Grid tool was developed in 2.14.21 and 2.18.15. One major difference between these two releases was found while using the QGIS Python API. This difference concerns the way a user define a layer type. The code block below compares how each QGIS release define a layer type.

```
#QGIS version 2.14.x
For layer in layers:
    If isinstance(layer, QgsRasterLayer):
        print layer.name()

#QGIS version 2.18.x
For layer in layers:
    If layer == QgsRasterLayer():
        print layer.name()
```

The 2.14.x method in defining a layer type will work in 2.18.x. The 2.18.x method will not work in 2.14. The 2.14.x method was used during tool development.

### 6.2.2 QGIS version 3.0 and Python 3.0

According to the release schedule, QGIS version 3.0 will be released in February 2018. This release will utilize, among other things:

- Qt 5 and PyQt 5 API
- Python 3.0

In general there are only a few differences between Python 3.0 and 2.7 versions. These differences are compared below. The 'print ('string')' method has been used during tool development.

Python 2.7		Python 3.0
input ()	→	raw_input ()
print "string"	→	print ("string")

The major difference will lies in the switch to the new APIs of PyQt5 and QGIS 3.0. Moreover, functions which previously relies on the packages from the OSGeo4W will not be usable as well, because OSGeo4W is not optimized for Python 3.x and Qt5 yet.

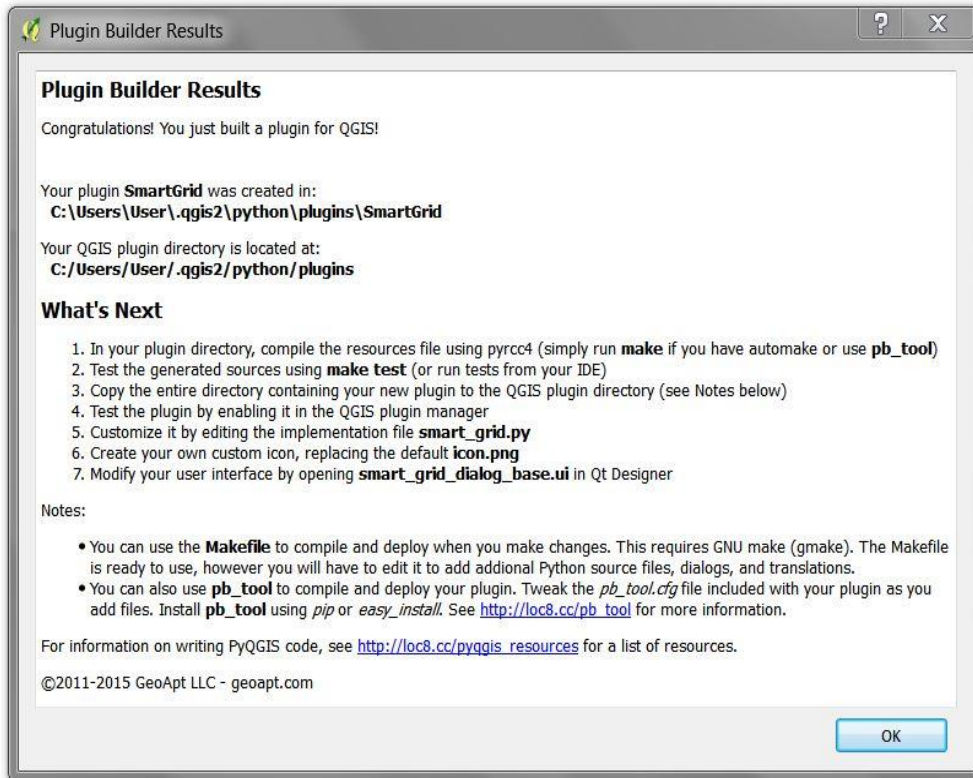
Although QGIS 3.0 will be released very soon, the 2.18.x version will still be maintained until September 2018. For future use of the Smart Grid plugin, it is recommended to follow the development of QGIS 3.0 after release.

## References

- Dummies. (2017). *Dummies*. Retrieved from Dummies:  
<http://www.dummies.com/programming/python/how-to-create-a-constructor-in-python/>
- eLeaf. (2017). *eLeaf Feed the world*. Retrieved from eleaf: <http://www.eleaf.com/>
- Jedlovec, G. (2009). *Automated Detection of Clouds in Satellite Imagery*. NASA .
- NOAA. (2017). *National Ocean Service*. Retrieved from oceanservice.noaa.gov:  
<https://oceanservice.noaa.gov/facts/remotesensing.html>
- Open Source Geospatial Foundation. (2017). *OSGeo4W*. Retrieved from trac.osgeo.org:  
<https://trac.osgeo.org/osgeo4w/>
- OSGeo project. (2017). *gdal*. Retrieved from gdal: <http://www.gdal.org/>
- PyQt Sourceforge. (2017). *PyQt reference guide*. Retrieved from PyQt reference guide:  
<http://pyqt.sourceforge.net/Docs/PyQt4/modules.html>
- Python Software Foundation. (2017). *Python Software Foundation*. Retrieved from Python Software Foundation: <https://docs.python.org/3/library/enum.html>
- QGIS . (2017). *QGIS* . Retrieved from QGIS : [www.qgis.org](http://www.qgis.org)
- QGIS. (2017). *QGIS*. Retrieved from QGIS:  
<https://www.qgis.org/en/site/getinvolved/development/roadmap.html#location-of-prereleases-nightly-builds>
- Qt. (2017). *QT*. Retrieved from QT: <https://www.qt.io/>
- Vallecha, A. (2015, September 20). *Quora*. Retrieved from Quora: <https://www.quora.com/What-are-constructors-in-Python>

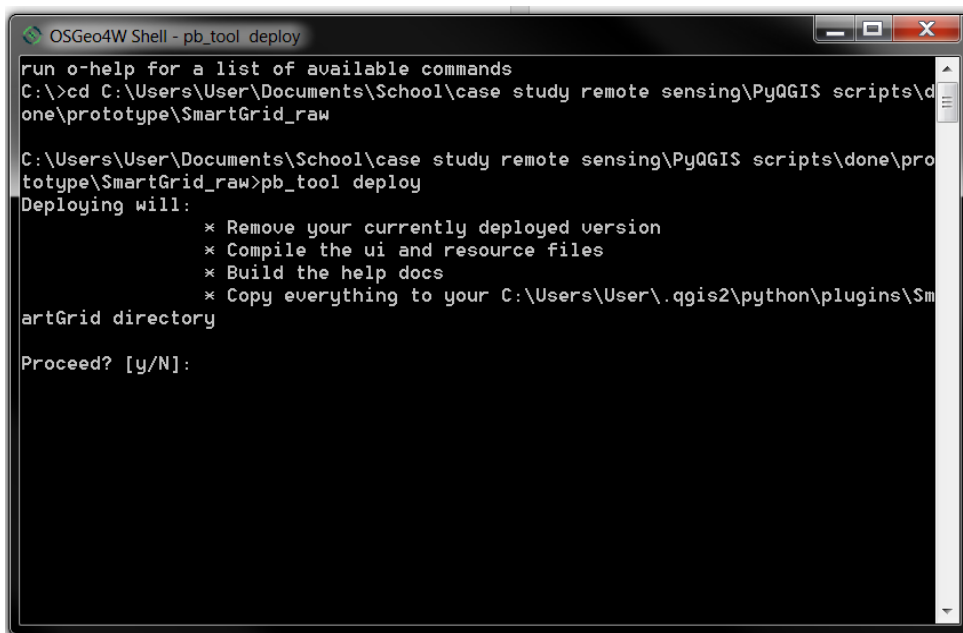
## Appendix 1 Screen dumps during development

### 1.1 Plugin Builder results





















1.2

### Compiling and deploying the plugin




## QGIS Cloud Masking Assistance Tool

### 1.3 Before and after compilation

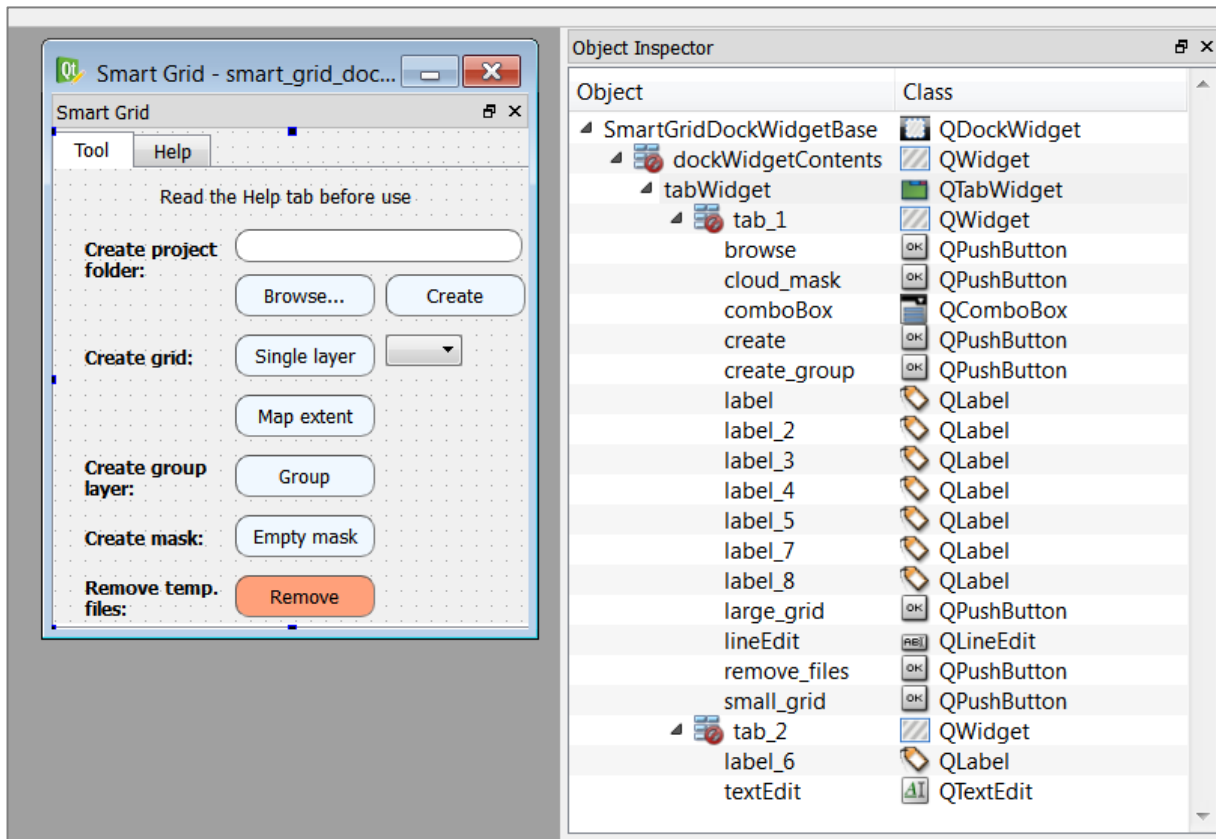
 help	3-1-2018 15:27	Bestandsmap	
 i18n	3-1-2018 15:27	Bestandsmap	
 scripts	3-1-2018 15:27	Bestandsmap	
 test	3-1-2018 15:27	Bestandsmap	
 pylintrc	29-12-2017 17:05	Bestand	9 kB
 smart_grid	30-12-2017 14:00	Python File	8 kB
 Makefile	30-12-2017 13:58	Bestand	8 kB
 resources	30-12-2017 14:51	Python File	6 kB
 plugin_upload	29-12-2017 17:05	Python File	4 kB
 pb_tool.cfg	30-12-2017 13:58	CFG-bestand	3 kB
 smart_grid_dockwidget	30-12-2017 13:58	Python File	2 kB
 README	30-12-2017 13:58	Firefox HTML Doc...	2 kB
 __init__	30-12-2017 13:58	Python File	2 kB
 icon	29-12-2017 17:05	PNG-afbeelding	2 kB
 README	30-12-2017 13:58	Tekstdocument	1 kB
 metadata	30-12-2017 13:58	Tekstdocument	1 kB
 smart_grid_dockwidget_base	30-12-2017 13:58	qt	1 kB
 resources.qrc	30-12-2017 13:58	QRC-bestand	1 kB

*Before compilation (lots of unnecessary files)*

 help	17-1-2018 9:49	Bestandsmap	
 __init__	17-1-2018 9:49	Python File	2 kB
 icon	17-1-2018 9:49	PNG-afbeelding	2 kB
 metadata	17-1-2018 9:49	Tekstdocument	1 kB
 resources	17-1-2018 9:49	Python File	6 kB
 smart_grid	17-1-2018 9:49	Python File	8 kB
 smart_grid_dockwidget	17-1-2018 9:49	Python File	2 kB
 smart_grid_dockwidget_base	17-1-2018 9:49	qt	1 kB

*After compilation (only essential files)*

## 1.4 GUI development in Qt Designer



On the left side, the actual plugin dialog (dock able widget) design is displayed. The objects/ elements of this dialog are shown on the right.