SEXTANTE for QGIS User's manual (v1.0)

Víctor Olaya

Edition 1.0 — Rev. March 27, 2012

SEXTANTE For QGIS User's manual

Edición 1.0
Rev. March 27, 2012

Contents

Chapter 1

Introduction

1.1 Introduction

Welcome to this introduction to SEXTANTE for QGIS. This text is targeted at those using SEXTANTE as their geospatial processing platform within QGIS

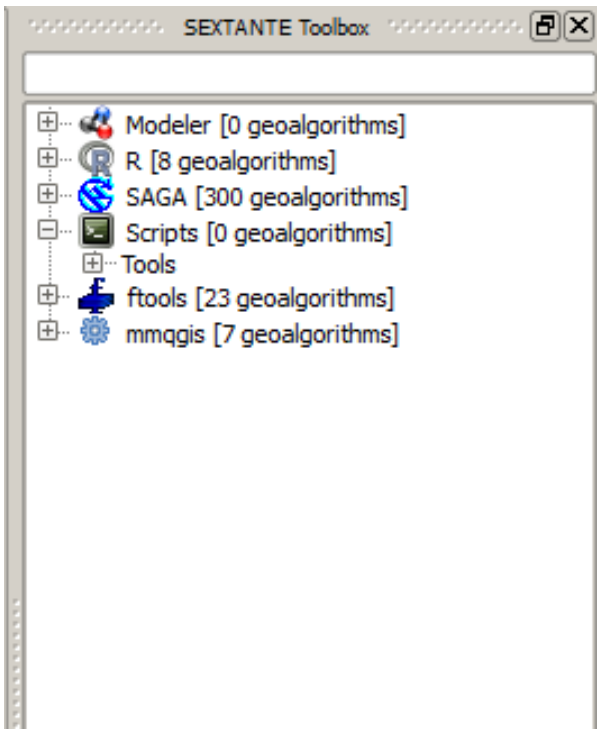
SEXTANTE is a geoprocessing environment that can be used to call native and third party algorithms from QGIS, making you spatial analysis tasks more productive and easy to accomplish.

In the following sections we will review how to use the graphical elements of SEXTANTE and take the most out of each one of them

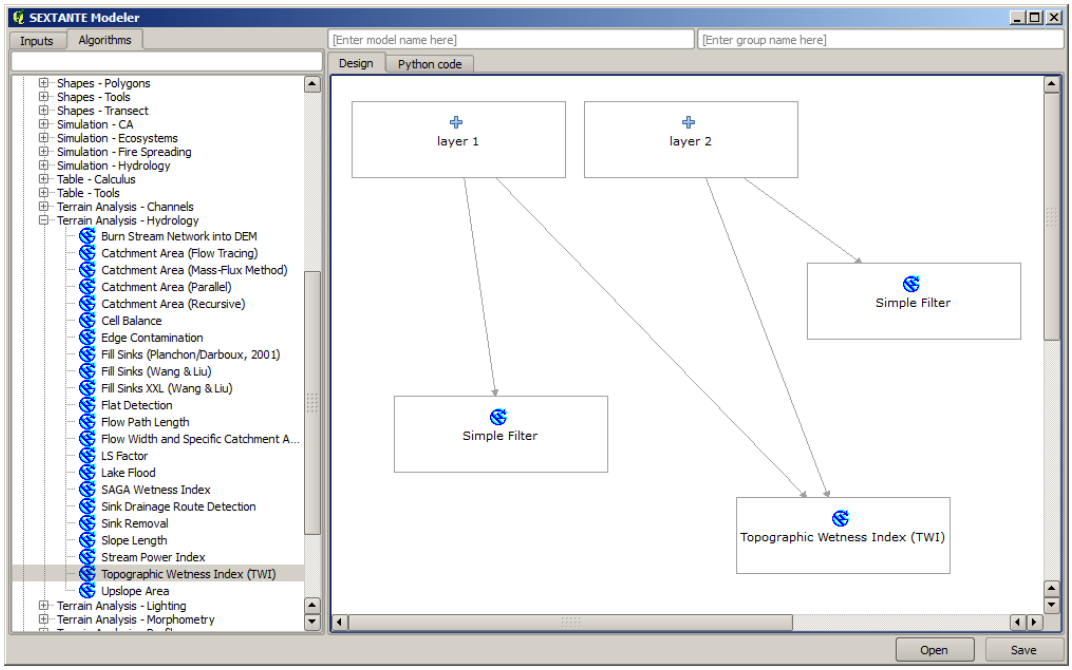
1.2 Basic elements of the SEXTANTE GUI

There are four basic elements in the SEXTANTE GUI, which are used to run SEXTANTE algorithms for different purposes. Chosing one tool or another will depend on the kind of analysis that is to be performed and the particular characteristics of each user and project. All of them (except for the batch processing interface, which is called from the toolbox, as we will see) can be accesed from the *SEXTANTE* menu item (you will see more than four entries. The remaining ones are not used to execute algorithms).

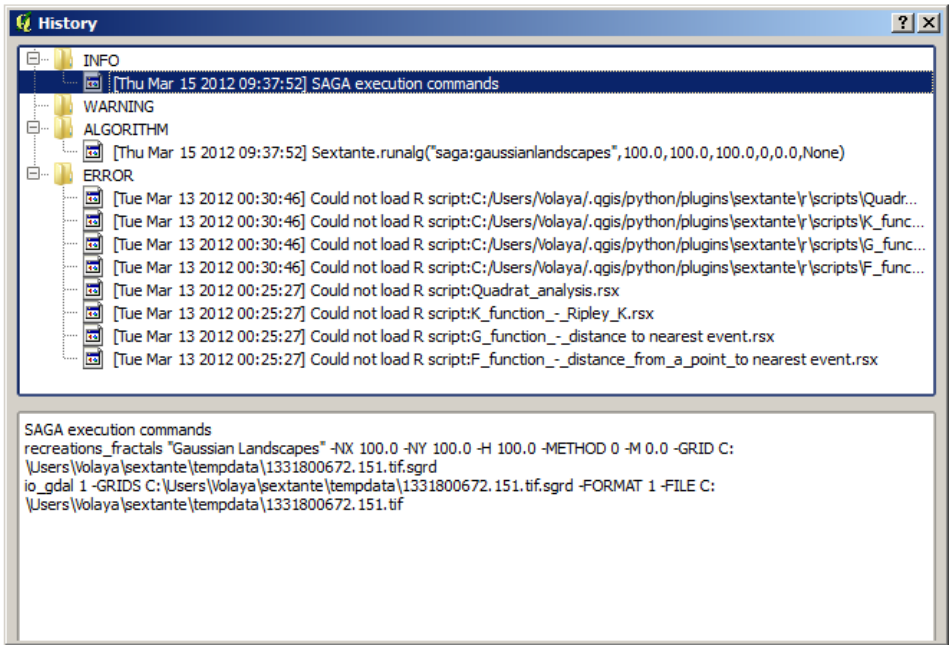
- The SEXTANTE toolbox. The main element of the SEXTANTE GUI, it is used to execute a single algorithm or run a batch process based on that algorithm.



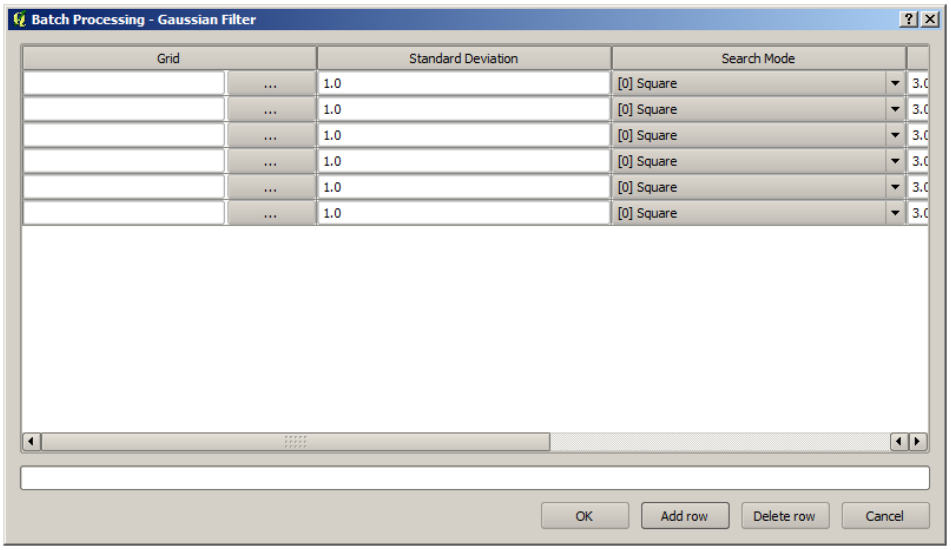
- The SEXTANTE graphical modeler. Several algorithms can be combined graphically using the modeler to define a workflow, creating a single process that involves several sub–processes



- The SEXTANTE history manager. All actions performed using any of the aforementioned elements are stored in a history file and can be later easily reproduced using the history manager



- The SEXTANTE batch processing interface manager. This interface allows you to execute batch processes and automate the execution of a single algorithm on multiple datasets.

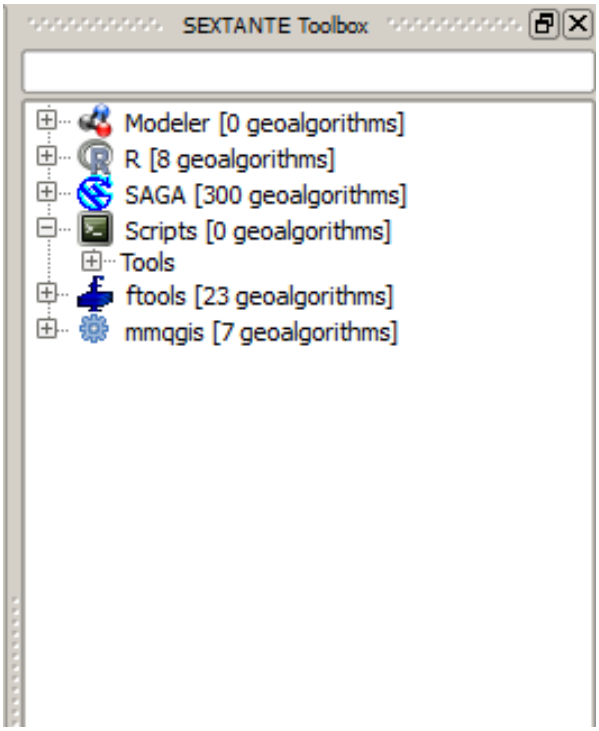


Along the following chapters we will review each one of this elements in detail.

The SEXTANTE toolbox

2.1 Introduction

The *Toolbox* is the main element of the SEXTANTE GUI, and the one that you are more likely to use in your daily work. It shows the list of all available algorithms grouped in different blocks, and is the access point to run them whether as a single process or as a batch process involving several executions of a same algorithm on different sets of inputs.



The toolbox contains all the algorithms available, divided into groups. Each group represents a so–called algorithm provider, which is a set of algorithms coming from the same source, for instance, from a third–party application with geoprocessing capabilities. Some of this groups represent algorithms from one of such third–party applications (currently, SAGA and R), while other contain algorithms directly coded along with SEXTANTE elements, not relying on any additional software. Currently, these providers all reuse code from already–existing QGIS plugins (more specifically, from the fTools vector library shiped along with

QGIS and the contributed mmqgis plugin that you can install using the plugin manager), making them more useful, since they can be executed from elements such as the modeler or the batch processing interface, which we will soon describe.

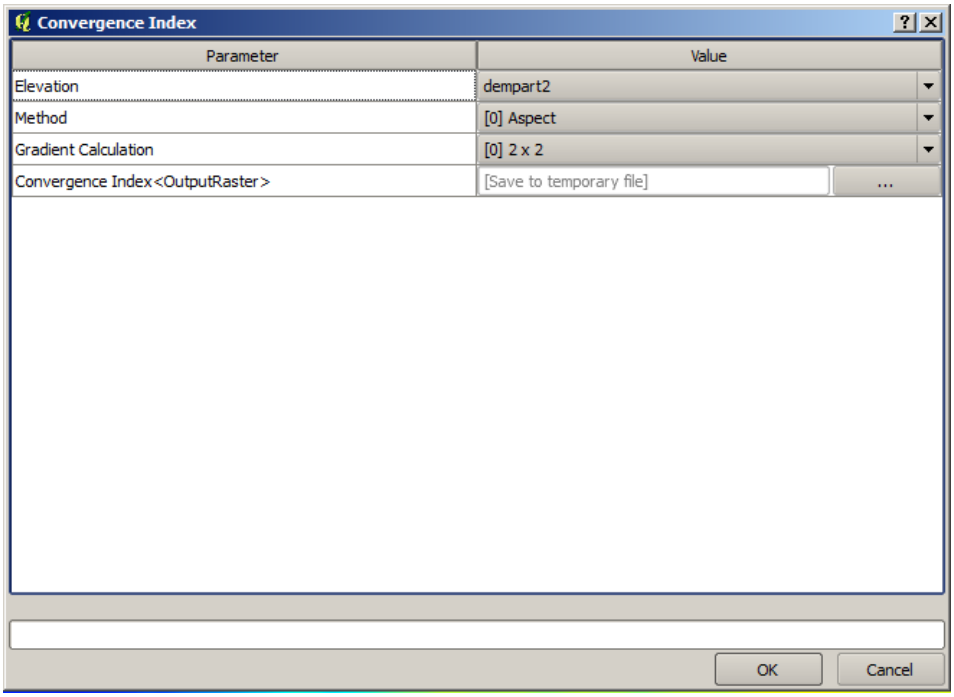
Additionaly, two more providers can be found, namely "Models" and "Scripts". This providers include user–created algorithms, and allow you to define your own workflows and processing tasks. We will devote a full chapter to each one of them a bit later.

In the upper part of the toolbox you can find a text box. To reduce the number of algorithms shown in the toolbox and make it easier to find the one you need, you can enter any word or phrase on the text box. Notice that, as you type, the number of algorithms in the toolbox is reduced to just those which contain the text you have entered in their names.

To execute an algorithm, just double–click on its name in the toolbox.

2.2 The algorithm dialog

Once you double–click on the name of the algorithm that you want to execute, a dialog similar to the next one is shown (in this case, the dialog corresponds to the *SAGA–Convergence index* algorithm).



This dialog is used to set the input values that the algorithm needs to be executed. It shows a table where input values and configuration parameters are to be set. It, of course, has a different content depending on the requirements of the algorithm to be executed, and is created automatically based on those requirements. On the left side, the name of the parameter is shown. On the right side the value of the parameter can be set.

Although the number and type of parameters depend on the characteristics of the algorithm, the structure is similar for all of them. The parameters found on the table can be of one of the following types.

- A raster layer, to select from a list of all the ones available (currently opened) in QGIS
- A vector layer, to select from a list of all the ones available in the QGIS

- A table, to select from a list of all the ones available in QGIS

- An option, to choose from a selection list of possible options

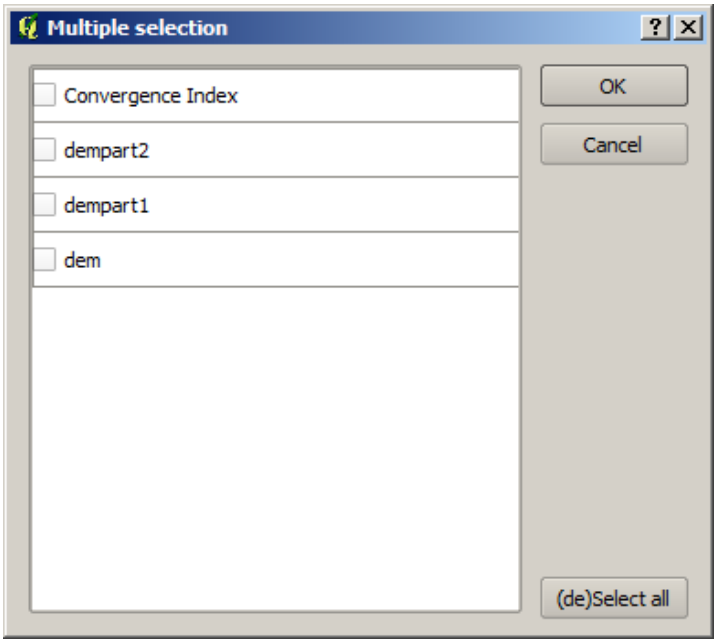
- A numerical value, to be introduced in a text box.

- A range, with min and max values to be introduced in two text boxes

- A text string, to be introduced in a text box

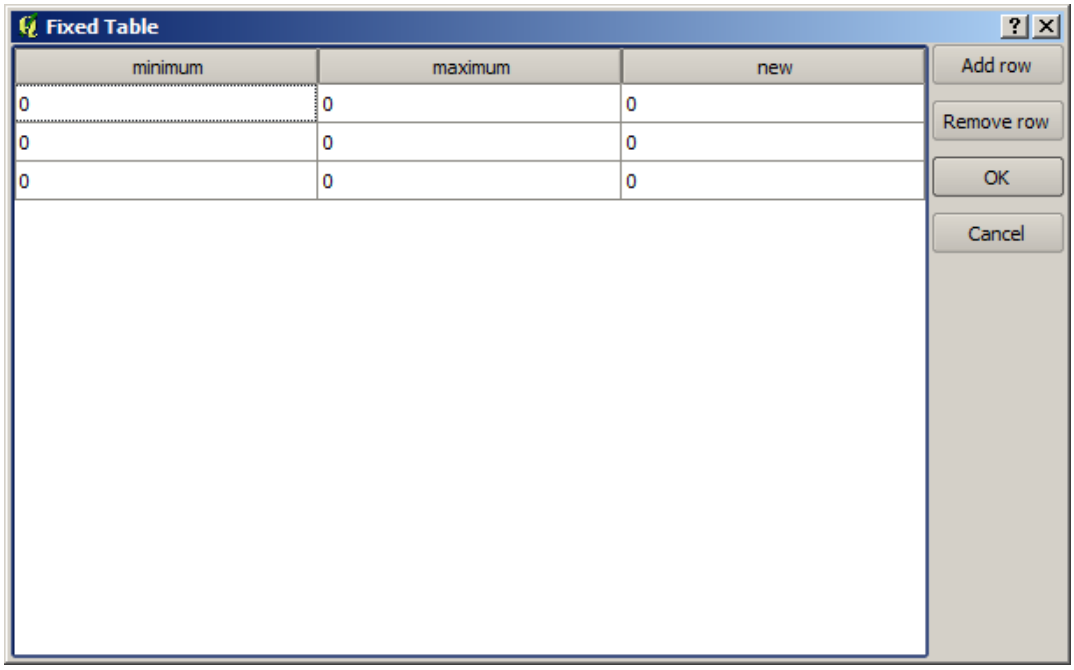
- A field, to choose from the attributes table of a vector layer or a single table selected in another parameter.

- A list of elements (whether raster layers, vector ones or tables), to select from the list of the ones available in QGIS. To make the selection, click on the small button on the left side of the corresponding row to see a dialog like the following one.



- A small table to be edited by the user. These are used to define parameters like lookup tables or convolution kernels, among others.

  Click on the button on the right side to see the table and edit its values.



Depending on the algorithm, the number of rows can be modified or not, using the buttons on the right side of the window.

You will find a help button in the lower part of the parameters dialog. If a help file is available, it will be shown, giving you more information about the algorithms and detailed descriptions of what each parameter does. Unfortunatley, most algorithms lack good documentation, but if you feel like contributing to the project, this would be a good place to start...

2.3 Data objects generated by SEXTANTE algorithms

Data objects generated by SEXTANTE can be of any of the following types:

- A raster layer
- A vector layer
- A table
- An HTML file (used for text and graphical outputs)

They are all saved to disk (there are no in–memory results), and the parameters table will contain a text box corresponding to each one of these outputs, where you can type the output channel to use for saving it. An output channel contains the information needed to save the resulting object somewhere. In the most usual case, you will save it to a file, but the architecture of SEXTANTE allows for any other way of storing it. For instance, a vector layer can be stored in a database or even uploaded to a remote server using a WFS–T service. Although solutions like these are not yet implemented, SEXTANTE is prepared to handle them, and we expect to add new kinds of output channels in a near feature.

To select an output channel, just click on the button on the right side of the text box. That will open a save file dialog, where you can select the desired filepath. Supported file extensions are shown in the file format selector of the dialog, depending on the kind of output and the algorithm.

The format of the output is defined by the filename extension. The supported formats depend on the ones supported by the algorithm itself. To select a format, just select the corresponding file extension (or add it if you are directly typing the filepath instead). If the extension of the filepath you entered does not match any of the supported ones, a default extension (usually `dbf` for tables, `tif` for raster layers and `shp` for vector ones) will be appended to the filepath and the file format corresponding to that extension will be used to save the layer or table.

If you do not enter any filename, the result will be saved as a temporary file and in the corresponding default file format, and will be deleted once you exit QGIS (take care with that in case you save your project and it contains temporary layers)

You can set a default folder for output data objects. Go to the configuration dialog (you can open it from the SEXTANTE menu), and in the "General" group you will find a parameter named "Output folder". This output folder is used as the default path in case you type just a filename with no path (i.e. `myfile.shp`) when executing an algorithm.

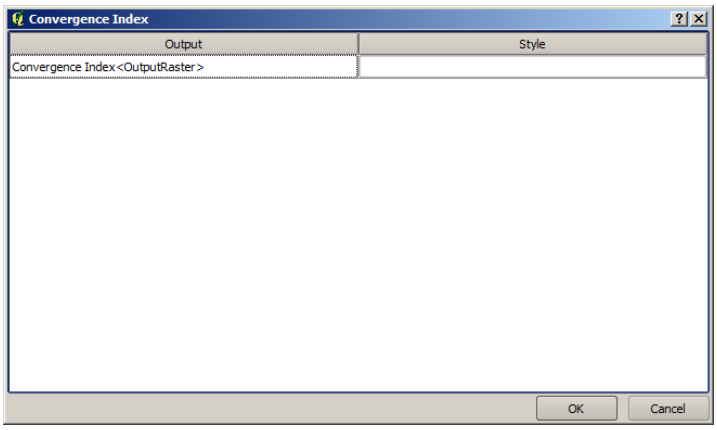
Apart from raster layers and tables, SEXTANTE also generates graphics and texts as HTML files. These results are shown at the end of the algorithm execution in a new dialog. This dialog will keep the results produced by SEXTANTE during the current session, and can be shown at any time by selecting the *SEXTANTE results viewer* menu.

2.4 Configuring SEXTANTE

As it has been mentioned, the configuration menu gives access to a new dialog where you can configure how SEXTANTE works. Configuration parameters are structured in separate blocks that you can select on the left–hand side of the dialog.

Along with the aforementioned "Output folder" entry, the "General" block contains parameters for setting the default rendering style for SEXTANTE layers (that is, layers generated by using algorihtms from any of the SEXTANTE components). Just create the style you want using QGIS, save it to a file, and then enter the path to that file in the settings so SEXTANTE can use it. Whenever a layer is loaded by SEXTANTE and added to the QGIS canvas, it will be rendered with that style.

Rendering stlyes can be configured individually for each algorithm and each one of its outputs. Just right–click on the name of the algorithm in the toolbox and select *Edit rendering styles*. You will see a dialog like the one shown next.



Select the style file (*.qml) that you want for each output and press OK.

Apart from the "General" block in the settigns dialog, you will also find one for each algorithm provider. They contain an "Activate" item that you can use to make algorithms

appear or not in the toolbox. Also, some algorithm providers have their own configuration items, that we will explain later when covering particular algorithm providers.

Chapter 3

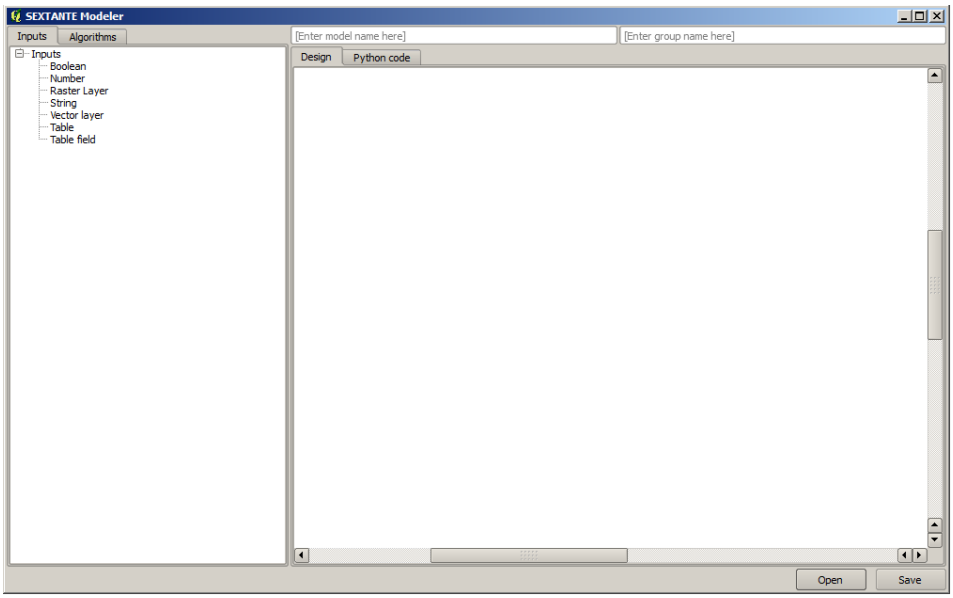
The SEXTANTE graphical modeler

3.1 Introduction

The *graphical modeler* allows to create complex models using a simple and easy–to–use interface. When working with a GIS, most analysis operations are not isolated, but part of a chain of operations instead. Using the graphical modeler, that chain of processes can be wrapped into a single process, so it is easier and more convenient to execute than a single process later on a different set on inputs. No matter how many steps and different algorithms it involves, a model is executed as a single algorithm, thus saving time and effort, specially for larger models.

The modeler can be opened from the SEXTANTE menu, but also from the toolbox. In the "Modeler" branch of the algorithms tree you will find a group named "tools", which contains an entry called "Create new model"

The modeler has a working canvas where the structure of the model and the workflow it represents are shown. On the left part of the window, a panel with two tabs can be used to add new elements to the model.



Creating a model involves two steps:

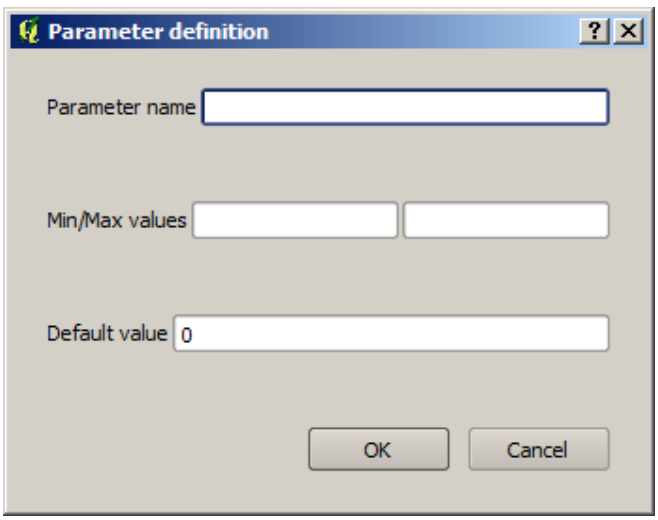
- *Definition of necessary inputs.* These inputs will be added to the parameters window, so the user can set their values when executing the model. The model itself is a SEXTANTE algorithm, so the parameters window is generated automatically as it happens with all the algorithms included in the library.
- *Definition of the workflow.* Using the input data of the model, the workflow is defined adding algorithms and selecting how they use those inputs or the outputs generated by other algorithms already in the model

3.2 Definition of inputs

The first step to create a model is to define the inputs it needs. The following elements are found in the *Inputs* tabs on the left side of the modeler window:

- Raster layer
- Vector layer
- String
- Table field
- Table
- Numerical value
- Boolean value

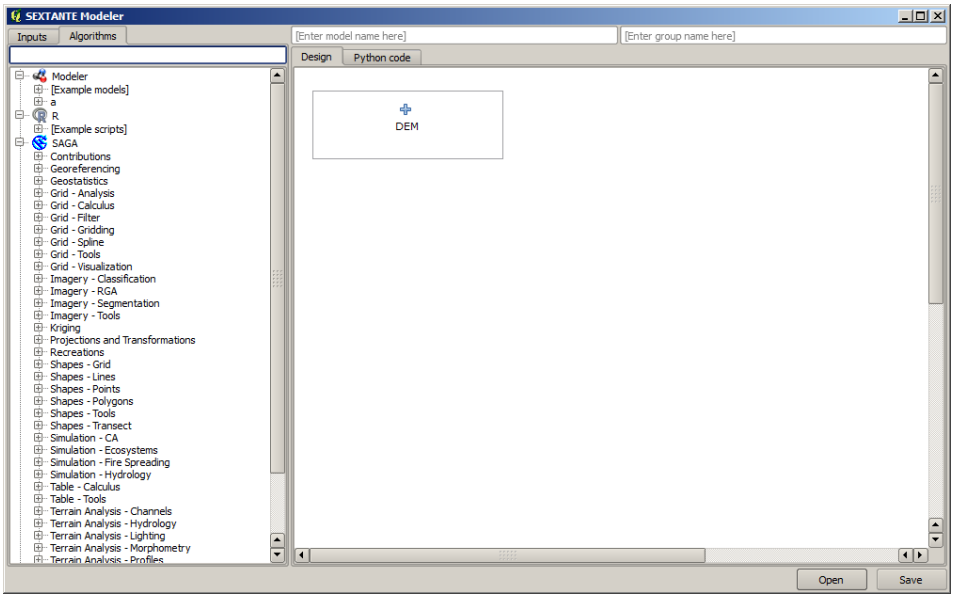
Double–clicking on any of them, a dialog is shown to define its caracteristics. Depending on the parameter itself, the dialog will contain just one basic element (the description, which is what the user will see when executing the model) or more of them. For instance, when adding a numerical value, as it can be seen in the next figure, apart from the description of the parameter you have to set a default value and a range of valid values.



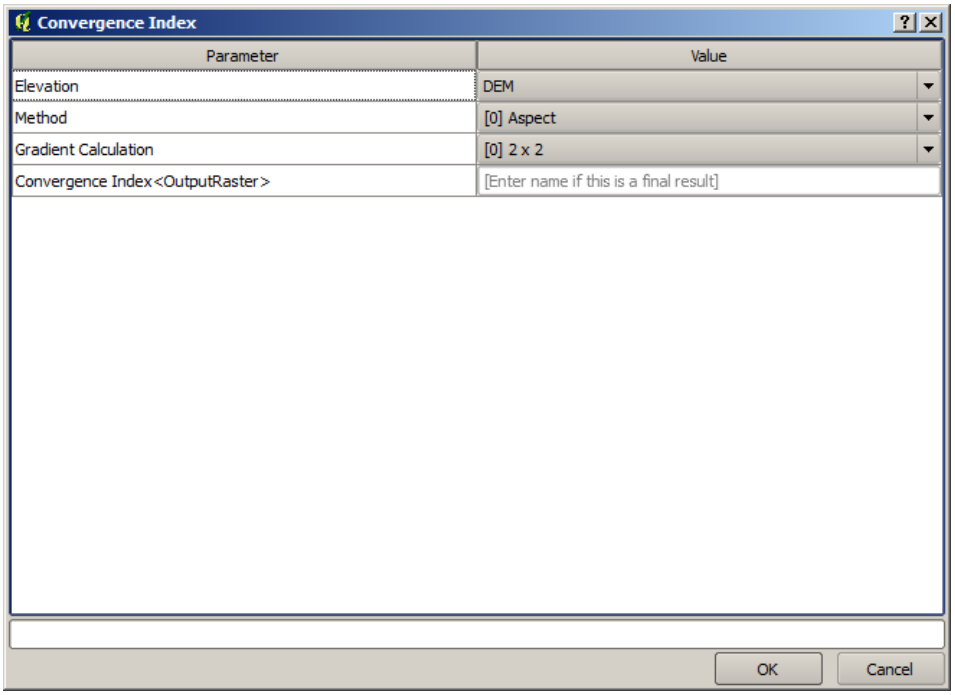
For each added input, a new element is added to the modeler canvas.

3.3 Definition of the workflow

Once the inputs have been defined, it is time to define the algorithms to apply on them. Algorithms can be found in the *Algorithms* tab, grouped much in the same way as they are in the toolbox.



To add a process, double–click on its name. An execution dialog will appear, with a content similar to the one found in the execution panel that SEXTANTE shows when executing the algorithm from the toolbox. the one shown next correspond to the SAGA convergence index algorithm, the same one we saw in the chapter dedicated to the SEXTANTe toolbox.



as you can see, some differences exist. Instead of the file output box that was used to set the filepath for output layers and tables, a simple text box is. If the layer generated by the

algorithm is just a temporary result that will be used as the input of another algorithm and should not be kept as a final result, just do not edit that textbox. Typing anything on it means that the result is a final one, and the text that you supply will be the description for the output, which will be the one the user will see when executing the model.

Selecting the value of each parameter is also a bit different, since there are importante differences between the context of the modeler and the toolbox one. Let's see how to introduce the values for each type of parameter.

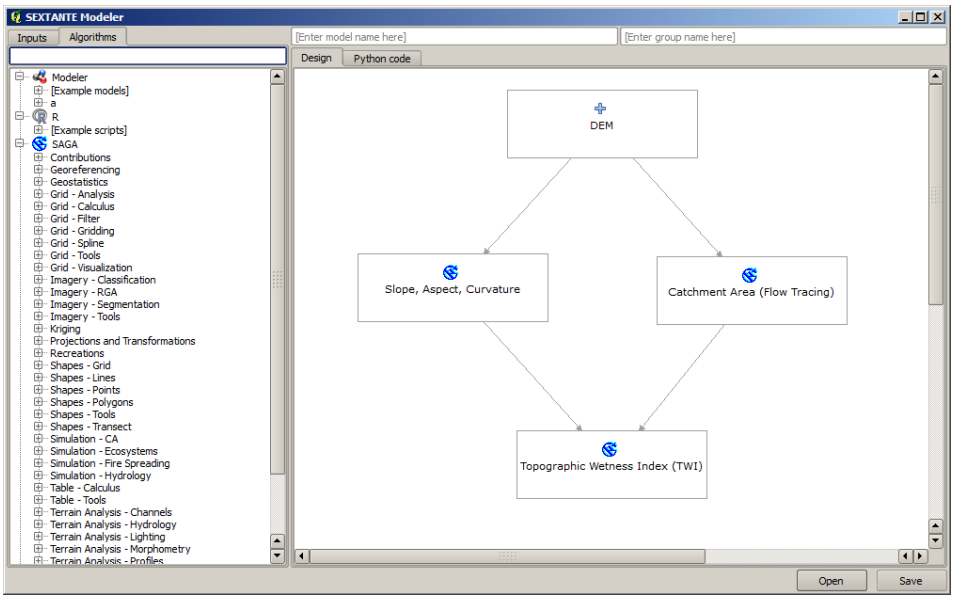
- Layers (raster and vector) and tables. They are selectend from a list, but in this case the possible values are not the layers or tables currently loaded in QGIS, but the list of model inputs of the corresponding type, or other layers or tables generated by algorithms already added to the model.

- Numerical values. Literal values can be introduced directly on the textbox. But this textbox is also a list that can be used to select any of the numerical value inputs of the model. In this case, the parameter will take the value introduced by the user when executing the model.

- String. Like in the case of numerical values, literal strings can be typed, or an input string can be selected.

- Table field. The fields of the parent table or layer cannot be known at design–time, since they depend of the selection of the user each time the model is executed. To set the value for this parameter, type the name of a field directly in the textbox, or use the list to select a table field input already added to the model. The validity of the selected field will be checked by SEXTANTE at run–time

Once all the parameter have been assigned valid values, click on *OK* and the algorithm will be added to the canvas. It will be linked to all the other elements in the canvas, whether algorithms or inputs, which provide objects that are used as inputs for that algorithm.



Elements can be dragged to a different position within the canvas, to change the way the module structure is displayed and make it more clear and intuitive. Links between elements are update automatically.

3.4 Saving and loading models

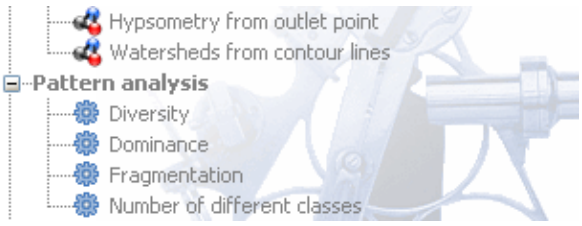
Use the *Save* button to save the current model and the *Open* one to open any model previously saved. Model are saved with the `.model` extension. If the model has been previously saved from the modeler window, you will not be prompted for a filename, since there is already a file associated with that model, and it will be used.

Before saving a model, you have to enter a name and a group for it, using the text boxes in the upper part of the window.

Models saved on the models folder (the default folder when you are prompted for a filename to save the model) will appear in the toolbox in the corresponding branch. When the toolbox is invoked, SEXTANTE searches the models folder for files with `.model` extension and loads the models they contain. Since a model is itself a SEXTANTE algorithm, it can be added to the toolbox just like any other algorithm.

The models folder can be set from the SEXTANTE configuration dialog, under the "Modeler" group.

Models loaded from the models folder appear not only in the toolbox, but also in the algorithms tree in the *Algorithms* tab of the modeler window. That means that you can incorporate a model as a part of a bigger model, just as you add any other algorithm.



3.5 SEXTANTE models as Python code

Along with the tab that contains the graphical design of the model, you will find another one containing a Python script which performs the same task as the model itself. Using that code, you can create a console script (we will explain them later in this same manual) and modify it to incorporate actions and methods not available in the graphical modeler, such as loops or conditional sentences.

This feature is also a very practical way of learning how to use SEXTANTE from the console and how to create SEXTANTE algorithms using Python code, so you can use it as a learning tool when you start creating your own SEXTANTE scripts.

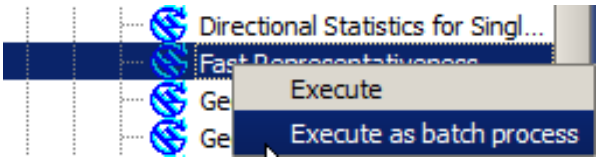
You will find a button below the text field containing the Python code. Click on it to directly create a new script from that code, without having to copy and paste it in the SEXTANTE script editor.

The SEXTANTE batch processing interface

4.1 Introduction

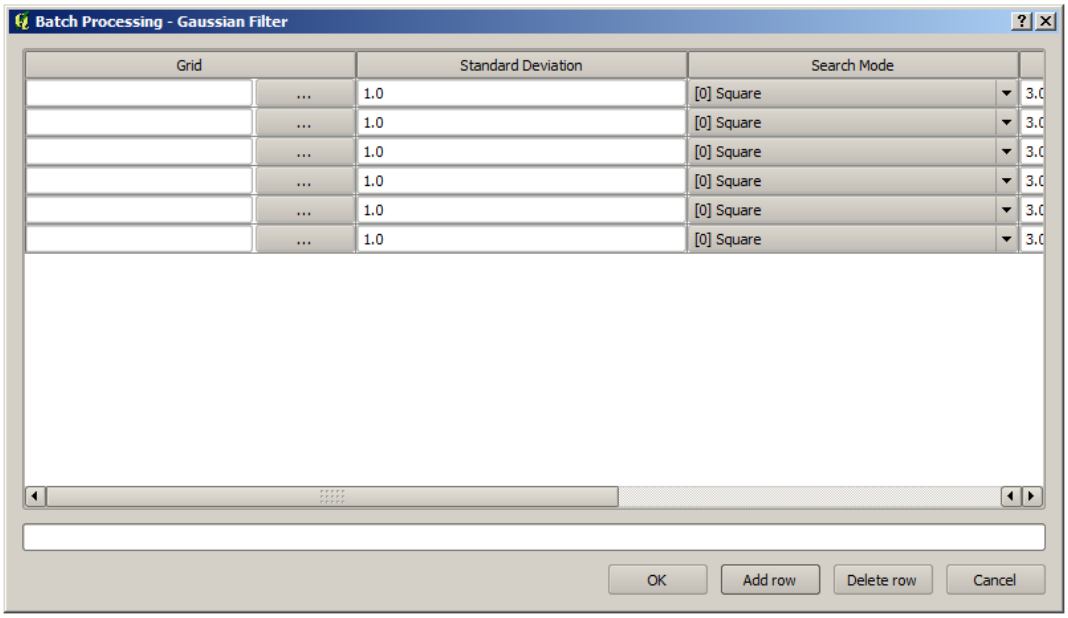
SEXTANTE algorithms (including models) can be executed as a batch process. That is, they can be executed using not a single set of inputs, but several of them, executing the algorithm as many times as needed. This is useful when processing large amounts of data, since it is not necessary to launch the algorithm many times from the toolbox.

To execute an algorithm as a batch process, right–click on its name in the toolbox and select the *Execute as batch process* option in the pop–up menu that will appear



4.2 The parameters table

Executing a batch process is similar to performing a single execution of an algorithm. Parameter values have to be defined, but in this case we need not just a single value for each parameter, but a set of them instead, one for each time the algorithm has to be executed. Values are introduced using a table like the one shown next.



Each line of this table represents a single execution of the algorithm, and each cell contains the value of one of the parameters. It is similar to the parameters dialog that you see when executing an algorithm from the toolbox, but with a different arrangement.

By default, the table contains just two rows. You can add or remove rows using the buttons on the lower part of the window.

Once the size of the table has been set, it has to be filled with the desired values

4.3 Filling the parameters table

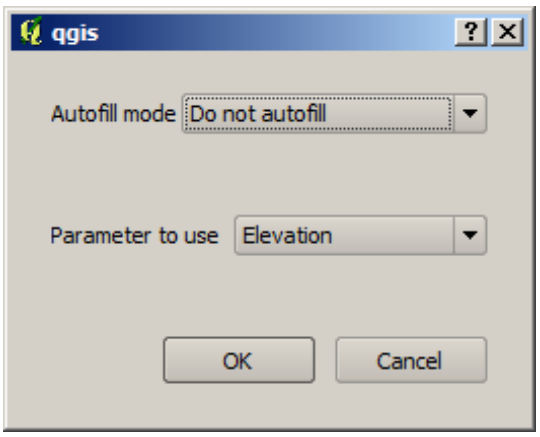
For most parameters, setting its value is trivial. Just type the value or select it from the list of available options, depending on the parameter type.

The main differences are found for parameters representing layers or tables, and for output filepaths. Regarding input layers and tables, when an algorithm is executed as part of a batch process those input data objects are taken directly from files, and not from the set of them already opened in QGIS. For this reason, any algorithm can be executed as a batch process even if no data objects at all are opened and the algorithm cannot be run from the toolbox.

Filenames for input data objects are introduced directly typing or, more conveniently, clicking on the button on the right hand of the cell, which shows a typical file chooser dialog. Multiple files can be selected at once. If the input parameter represents a single data object and several files are selected, each one of them will be put in a separate row, adding new ones if needed. If it represents a multiple input, all the selected files will be added to a single cell, separated by semicolons.

Output data objects are always saved to a file and, unlike when executing an algorithm from the toolbox, saving to a temporary one is not permitted. You can type the name directly or use the file chooser dialog that appears when clicking on the accompanying button.

Once you select the file, a new dialog is shown to allow for autocompletion of other cells in the same column (same parameter).



If the default value (*Do not autocomplete*) is selected, SEXTANTE will just put the selected filename in the selected cell from the parameters table. If any of the other options is selected, all the cells below the selected one will be automatically filled based on a defined criteria. This way, it is much easier to fill the table, and the batch process can be defined with less effort.

Automatic filling can be done simply adding correlative numbers to the selected filepath, or appending the value of another field at the same row. This is particularly useful for naming output data object according to input ones.

Slope
C:\Documents and Settings\usuario\Mis documentos\slope1.tif
C:\Documents and Settings\usuario\Mis documentos\slope2.tif
C:\Documents and Settings\usuario\Mis documentos\slope3.tif
C:\Documents and Settings\usuario\Mis documentos\slope4.tif

4.4 Executing the batch process

To execute the batch process once you have introduced all the necessary values, just click on *OK*. SEXTANTE will show the progress of the global batch process in the progress bar in the lower part of the dialog.

Chapter 5

Using SEXTANTE from the console. Creating scripts.

5.1 Introduction

The console allows advanced users to increase their productivity and perform complex operations that cannot be performed using any of the other elements of the SEXTANTE GUI. Models involving several algorithms can be defined using the command–line interface, and additional operations such as loops and conditional sentences can be added to create more flexible and powerful workflows.

There is not a SEXTANTE console in QGIS, but all SEXTANTE commands are available instead from QGIS built–in Python console. That means that you can incorporate those command to your console work and connect SEXTANTE algorithms to all the other features (including methods from the QGIS API) available from there.

The code that you can execute from the Python console, even if it does call any SEXTANTE method, can be converted into a new SEXTANTE algorithm that you can later call from the toolbox, the graphical modeler or any other SEXTANTE component, just like you do with any other SEXTANTE algorithm. In fact, some algorithms that you can find in the toolbox, like all the ones in the *mmqgis* group, are simple scripts.

In this chapter we will see how to use SEXTANTE from the QGIS Python console, and also how to write your own algorithms using Python.

5.2 Calling SEXTANTE from the Python console

The first thing you have to do is to import the `Sextante` class with the following line:

```
>>from sextante.core.Sextante import Sextante
```

Now, there is basically just one (interesting) thing you can do with SEXTANTE from the console: to execute an algorithm. That is done using the `runalg()` method, which takes the name of the algorithm to execute as its first parameter, and then a variable number of additional parameter depending on the requirements of the algorithm. So the first thing you need to know is the name of the algorithm to execute. That is not the name you see in the toolbox, but rather a unique command–line name. To find the right name for your algorithm, you can use the `algslist()` method. Type the following line in you console:

You will see something like this.

```
>>> Sextante.alglist()
Accumulated Cost (Anisotropic)---------------->saga:accumulatedcost(anisotropic)
Accumulated Cost (Isotropic)------------------>saga:accumulatedcost(isotropic)
Add Coordinates to points--------------------->saga:addcoordinatestopoints
Add Grid Values to Points--------------------->saga:addgridvaluestopoints
Add Grid Values to Shapes--------------------->saga:addgridvaluestoshapes
Add Polygon Attributes to Points-------------->saga:addpolygonattributestopoints
Aggregate------------------------------------->saga:aggregate
Aggregate Point Observations------------------>saga:aggregatepointobservations
Aggregation Index----------------------------->saga:aggregationindex
Analytical Hierarchy Process------------------>saga:analyticalhierarchyprocess
Analytical Hillshading------------------------>saga:analyticalhillshading
Average With Mask 1--------------------------->saga:averagewithmask1
Average With Mask 2--------------------------->saga:averagewithmask2
Average With Thereshold 1--------------------->saga:averagewiththereshold1
Average With Thereshold 2--------------------->saga:averagewiththereshold2
Average With Thereshold 3--------------------->saga:averagewiththereshold3
B-Spline Approximation------------------------>saga:b-splineapproximation
.
.
.
```

That's a list of all the available algorithms, alphabetically ordered, along with their corresponding command–line names.

You can use a string as a parameter for this method. Instead of returning the full list of algorithm, it will only display those that include that string. If, for instance, you are looking for an algorithm to calculate slope from a DEM, type `alglist("slope")` to get the following result:

```
DTM Filter (slope-based)---------------------->saga:dtmfilter(slope-based)
Downslope Distance Gradient------------------->saga:downslopedistancegradient
Relative Heights and Slope Positions---------->saga:relativeheightsandslopepositions
Slope Length---------------------------------->saga:slopelength
Slope, Aspect, Curvature---------------------->saga:slopeaspectcurvature
Upslope Area---------------------------------->saga:upslopearea
Vegetation Index[slope based]----------------->saga:vegetationindex[slopebased]
```

It is easier now to find the algorithm you are looking for and its command–line name, in this case *saga:slopeaspectcurvature*

Once you know the command–line name of the algorithm, the next thing to do is to know the right syntax to execute it. That means knowing which parameters are needed and the order in which they have to be passed when calling the `runalg()` method. SEXTANTE has a method to describe an algorithm in detail, which can be used to get a list of the parameters that an algorithms require and the outputs that it will generate. To do it, you can use the `alghelp(name_of_the_algorithm)` method. Use the command–line name of the algorithm, not the full descriptive name.

Calling the method with `saga:slopeaspectcurvature` as parameter, you get the following description.

```
>Sextante.alghelp("saga:slopeaspectcurvature")
ALGORITHM: Slope, Aspect, Curvature
```

```
ELEVATION <ParameterRaster>
METHOD <ParameterSelection>
SLOPE <OutputRaster>
ASPECT <OutputRaster>
CURV <OutputRaster>
HCURV <OutputRaster>
VCURV <OutputRaster>
```

Now you have everything you need to run any algorithm. As we have already mentioned, there is only one single command to execute algorithms: `runalg`. Its syntax is as follows:

```
> runalg{name_of_the_algorithm, param1, param2, ..., paramN,
        Output1, Output2, ..., OutputN)
```

The list of parameters and ouputs to add depends on the algorithm you want to run, and is exactly the list that the `describealg` method gives you, in the same order as shown.

Depending on the type of parameter, values are introduced differently. The next one is a quick review of how to introduce values for each type of input parameter

- Raster Layer, Vector Layer or Table. Simply use a string with the name that identifies the data object to use (the name it has in the QGIS Table of Contents) or a filename (if the corresponding layer is not opened, it will be opened, but not added to the map canvas). If you have an instance of a QGIS object representing the layer, you can also pass it as parameter. If the input is optional and you do not want to use any data object, use `None`.

- Selection. If an algorithm has a selection parameter, the value of that parameter should be entered using an integer value. To know the available options, you can use the `algoptions` command, as shown in the following example:

  ```
  >>Sextante.algoptions("saga:slopeaspectcurvature")
  METHOD(Method)
  0 - [0] Maximum Slope (Travis et al. 1975)
  1 - [1] Maximum Triangle Slope (Tarboton 1997)
  2 - [2] Least Squares Fitted Plane (Horn 1981, Costa-Cabral & Burgess 1996)
  3 - [3] Fit 2.Degree Polynom (Bauer, Rohdenburg, Bork 1985)
  4 - [4] Fit 2.Degree Polynom (Heerdegen & Beran 1982)
  5 - [5] Fit 2.Degree Polynom (Zevenbergen & Thorne 1987)
  6 - [6] Fit 3.Degree Polynom (Haralick 1983)
  ```

  In this case, the algorithm has one of such such parameters, with 7 options. Notice that ordering is zero–based.

- Multiple input. The value is a string with input descriptors separated by semicolons. As in the case of single layers or tables, each input descriptor can be the data object name, or its filepath.

- Table Field from XXX. Use a string with the name of the field to use. This parameter is case–sensitive.

- Fixed Table. Type the list of all table values separated by commas and enclosed between quotes. Values start on the upper row and go from left to right. You can also use a 2D array of values representing the table.

Boolean, string and numerical parameters do not need any additional explanations.

Input parameters such as strings or numerical values have default values. To use them, `None` in the corresponding parameter entry.

For output data objects, type the filepath to be used to save it, just as it is done from the toolbox. If you want to save the result to a temporary file, use `None`. The extension of the file determines the file format. If you enter a file extension not included in the ones supportede by the algorithm, the default file format for that output type will be used, and its corresponding extension appended to the given filepath.

Unlike when an algorithm is executed from the toolbox, outputs are not added to the map canvas if you execute that same algorithm from the Python console. If you want to add an output to it, you have to do it yourself after running the algorithm. To do so, you can use QGIS API commands, or, even easier, use one of the handy methods provided by SEXTANTE for such task.

The `runalg` method returns a dictionary with the output names (the ones shown in the algorithm description) as keys and the filepaths of those outputs as values. To add all the outputs generated by an algorithm, pass that dictionary to the `loadFromAlg()` method. You can also load an individual layer passing its filepath to the `load()` method.

5.3 Creating scripts and running them from the toolbox

You can create your own algorithms by writing the corresponding Python code and adding a few extra lines to supply additional information needed by SEXTANTE. You can find a *Create new script* under the tools group in the script algorithms block of the toolbox. Double click on it to open the script edition dialog. That's where you should type your code. Saving the script from there in the scripts folder (the default one when you open the save file dialog), with `.py` extension, will automatically create the corresponding algorithm.

The name of the algorithm (the one you will see in the toolbox) is created from the filename, removing its extension and replacing low hyphens with blank spaces.

Let's have the following code, which calculates the Topographic Wetness Index(TWI) directly from a DEM

```
##dem=raster
##twi=output
ret_slope = Sextante.runalg("saga:slopeaspectcurvature", dem, 0, None,
                None, None, None, None)
ret_area = Sextante.runalg("saga:catchmentarea(mass-fluxmethod)", "dem",
                0, False, False, False, False, None, None, None, None, None)
Sextante.runalg("saga:topographicwetnessindex(twi), ret_slope['SLOPE'],
                ret_area['AREA'], None, 1, 0, twi)
```

As you can see, it involves 3 algorithms, all of them coming from SAGA. The last one of them calculates de TWI, but it needs a slope layer and a flow accumulation layer. We do not have these ones, but since we have the DEM, we can calculate them calling the corresponding SAGA algorithms.

The part of the code where this processing takes place is not difficult to understand if you have read the previous sections in this chapter. The first lines, however, need some additional explanation. They provide SEXTANTE the information it needs to turn your code into an algorithm that can be run from any of its components, like the toolbox or the graphical modeler.

These lines start with a double Python comment symbol and have the following structure: *[parameter_name]=[parameter_type] [optional_values]*. Here is a list of all the parameter types that SEXTANTE supports in its scripts, their syntax and some examples.

- `raster`. A raster layer
- `vector`. A vector layer
- `table`. A table
- `raster`. A numerical value. A default value must be provided. For instance, `depth=number 2.4`
- `string`. A text string. As in the case of numerical values, a default value must be added. For instance, `name=string Victor`
- `boolean`. A boolean value. Add `True` or `False` after it to set the default value. For example, `verbose=boolean True`
- `multiple raster`. A set of input raster layers.
- `multiple vector`. A set of input vector layers.
- `multiple table`. A set of input tables.
- `field`. A field in the attributes table of a vector layer. the name of the layer has to be added after the `field` tag. For instance, if you have declared a vector input with `mylayer=vector`, you could use `myfield=field mylayer` to add a field from that layer as parameter.

The parameter name is the name that will be shown to the user when executing the algorithm, and also the variable name to use in the script code. The value entered by the user for that parameter will be asigned to a variable with that name.

Layers and tables values are strings containing the filepath of the corresponding object. To turn them into a QGIS object, you can use the `getObject()` method in the `Sextante` class. Multiple inputs also have a string value, which contains the filepaths to all selected object, separated by semicolons.

Output are defined in a similar manner, using the following tags:

- `output raster`
- `output vector`
- `output table`

The value asigned to the output variables is always a string with a filepath. It will correspond to a temporary filepath in case the user has not entered any output filename.

When you declare an output, SEXTANTE will try to add it to QGIS once the algorithm is finished. That is the reason why, although the `runalg()` method does not load the layers it produces, the final TWI layer will be loaded, since it is saved to the file entered by the user, which is the value of the corresponding output.

Do not use the `load()` method in your scripta algorithms, but just when working with the console line. If a layer is created as output of an algorithm, it should be declared as such. Otherwise, you will not be able to properly use the algorithm in the modeler, since its syntax (as defined by the tags explained above) will not match what the algorithm really creates.

In addition to the tags for parameters and outputs, you can also define the group under which the algorithm will be shown, using the `group` tag.

Several examples are provided with SEXTANTE. Please, check them to see real examples of how to create algorithms using this feature of SEXTANTE. You can right–click on any script algorithm and select *Edit script* to edit its code or just to see it.

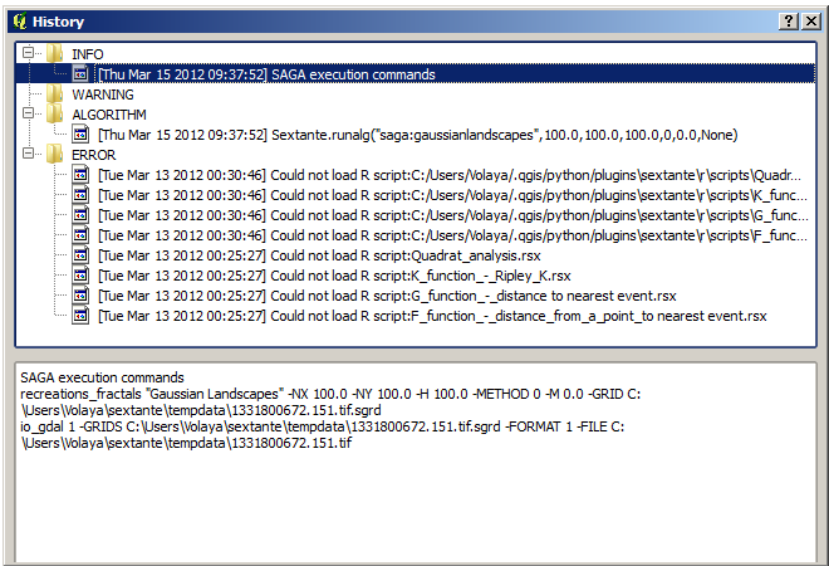
The SEXTANTE history manager

6.1 Introduction

Every time you execute a SEXTANTE algorithm, information about the process is stored in the SEXTANTE history manager. Along with the parameters used, the date and time of the execution are also saved.

This way, it is easy to track the and control all the work that has been developed using SEXTANTE, and easily reproduce it.

The SEXTANTE history manager is a set of registry entries grouped according to their date of execution, making it easier to find information about an algorithm executed at any particular moment.



Process information is kept as a command–line expression, even if the algorithm was launched from the toolbox. This makes it also useful for those learning how to use the command–lin interface, since they can call an algorithm using the toolbox and then check the history manager to see how that same algorithm could be called from the command line.

Apart from browsing the entries in the registry, processes can be re–executed, simply double–clicking on the corresponding entry.

Along with algorithm executions, SEXTANTE communicates with the user using the other groups of the registry, namely *Errors, Warnings* and *Information*. In case something is not

working properly, having a look at the *Errors* might help you to see what is happening. If you get in contact with a SEXTANTE developer to report a bug or error, the information in that group will be very useful for him to find out what is going wrong.

When executing third party algorithms, this is usually done calling their command–line interfaces, which communicate with the user using the console. Although that console is not shown, a full dump of it is stored in the *Information* group each time you run one of those algorithms. If, for instance, you are having problems executing a SAGA algorithm, look for en entry name *SAGA execution console output* to check all the messages generated by SAGA and try to find out where the problem is.

Some algorithms, even if they can produce a result with the given input data, might add comments or additional information to *Warning* in case they detect potential problems from that data, in order to warn you about them. Make sure you check those messages in case you are having unexpected results.

Chapter 7

Configuring external applications

7.1 Introduction

SEXTANTE can be extended using additional applications, calling them from within SEXTANTE. Currently, SAGA, GRASS and R are supported. This chapter will show you how to do it. Once you have configured the system, you will be able to execute external algorithms from any SEXTANTE component like the toolbox or the graphical modeler, just like you do with any other SEXTANTE geoalgorithm.

7.1.1 A note on file formats

When using an external software, opening a file in QGIS does not mean that it can be opened and processed as well on that other woftware. In most cases, it can read what you have opened in QGIS, but in some cases, that might not be the case. When using databases or uncommon file formats, whether for raster of vector layers, problems might arise. If that happens, try to use well known file formats that you are sure that are understood by both programs, and check to console output (in the history and log dialog) for knowing more about what is going wrong.

Currently, if the layer you want to process comes from a remote service or a database connection, it cannot be processed by algorithms coming from SGA, GRASS or R, but we are working on solving this issue...(stay tuned for the next version, which might probably be able to do it)

Regarding output formats, raster layers can be saved as TIFF (.tif) (the default format) or ESRI ASCII files (.asc), while vector layers are saved as shapefiles (.shp). These have been chosen as the *lingua franca* between supported third party applications and QGIS. Providers not using external application can process anylayer that you can open in QGIS, since they open it for analysis trough QGIS.

7.2 SAGA

SAGA algorithms can be run from SEXTANTE if you have SAGA installed in your system and you configure SEXTANTE properly so it can find SAGA executables. In particular, the SAGA command–line executable is needed to run SAGA algorithms. SAGA binaries are not included with SEXTANTE, so you have to download and install the software yourself. Please check the SAGA website at for more information.

Once SAGA is installed, open the SEXTANTE configuration dialog. In the *SAGA* block you will find a setting named *SAGA Folder*. Enter the path to the folder when saga is installed. Close the configuration dialog and now you are ready to run SAGA algorithms from SEXTANTE.

Notice that, ever before doing that, SAGA algorithms are shown in the toolbox and you can open them to fill the corresponding parameters dialog. However, if you try to run the algorithm after entering the parameter values, SEXTANTE will show an error message. This is because the algorithm descriptions (needed to create the parameters dialog and give SEXTANTE the information it needs about the algorithm) are not included with SAGA, but with SEXTANTE instead. That is, they are part of SEXTANTE, so you have them in yur installation even if you have not installed SEXTANTE. Running the algorithm, however, needs SAGA binaries installed in your system.

7.2.1 About SAGA grid system limitations

Most of SAGA algorithms that require several input raster layers, require them to have the same grid system. That is, to cover the same geographic area and have the same cellsize, so their corresponding grids match. When calling SAGA algorithms from SEXTANTE, you can use any layer, regarless of its cellsize and extent. When multiple raster layers are used as input for a SAGA algorithm, SEXTANTE resamples them to a common grid system and then passes them to SAGA.

The definition of that common grid system is controlled by the user, and you will find several parameters in the SAGA group of the setting window to do so. There are two ways of setting the target grid system:

- Setting it manually. You define the extent setting the values of the following parameters:
  - Resampling min X
  - Resampling max X
  - Resampling min Y
  - Resampling max Y
  - Resampling cellsize

  Notice that SEXTANTE will resample input layers to that extent, even if they do not overlap with it.

- Setting it automatically from input layers. To select this option, just check the "Use min covering grid system for resampling" option. All the other settings will be ignored and the minimum extent that covers all the input layers will be used. The cellsize of the target layer is the maximum of all cellsizes of the input layers.

For algorithms that do not use multiple raster layers, or for those that do not need a unique input grid system, no resampling is performed before calling SAGA, and those parameters are not used.

7.2.2 About vector layer selections

By default, when a SAGA algorithm takes a vector layer, it will use all its features, even if a selection exist in QGIS. You can make SAGA aware of that seelction by checking the *Use selected features* item in the SAGA settings group. When you do so, each time you execute a

SAGA algorithm that uses a vector layer, the selected features of that layer will be exported to a new layer, and SAGA will work with that new layer instead.

Notice that if you select this option, a layer with no selection will behave like a layer with all its features selected, not like an empty layer.

7.3 R. Creating R scripts

R integration in SEXTANTE is different from that of SAGA in that there is not a predefined set of algorithms you can run (except for a few examples). Instead, you should write your scripts and call R commands, much like you would do from R, and in a very similar manner to what we saw in the chapter dedicated to SEXTANTE scripts. This chapter shows you the syntax to use to call those R commands from SEXTANTE and how to use SEXTANTE objects (layers, tables) in them.

The first thing you have to do, as we saw in the case of SAGA, is to tell SEXTANTE where you R binaries are located. You can do so using the *R folder* entry in the SEXTANTE configuration dialog. Once you have set that parameter, you can start creating your own R scripts and executing them.

To add a new algorithm that calls an R function (or a more complex R script that you have developed and you would like to have available from SEXTANTE), you have to create a script file that tells SEXTANTE how to perform that operation and the corresponding R commands to do so.

Script files have the extension `rsx` and creating them is pretty easy if you just have a basic knowledge of R syntax and R scripting. They should be stored in the R scripts folder. You can set this folder in the R settings group (available from the SEXTANTE settings dialog), just like you do with the folder for regular SEXTANTE scripts.

Let's have a look at a very simple file script file, which calls the R method `spsample` to create a random grid within the boundary of the polygons in a given polygon layer. This method belong to the `maptools` package. Since almost all the algorithms that you might like to incorporate into SEXTANTE will use or generate spatial data, knowledge of spatial packages like `maptools` and, specially, `sp`, is mandatory.

```
##polyg=vector
##numpoints=number 10
##output=output vector
##sp=group
pts=spsample(polyg,numpoints,type="random")
output=SpatialPointsDataFrame(pts, as.data.frame(pts))
```

The first lines, which start with a double Python comment sign (##), tell SEXTANTE the inputs of the algorithm described in the file and the outputs that it will generate. They work exactly with the same syntax as the SEXTANTE scripts that we have already seen, so they will not be described here again. Check the corresponding chapter for more information.

When you declare an input parameter, SEXTANTE uses that information for two things: creating the user interface to ask the user for the value of that parameter and creating a corresponding R variable that can be later used as input for R commands

In the above example, we are declaring an input of type `vector polygon` named `polyg`. When executing the algorithm, SEXTANTE will open in R the layer selected by the user and store it in a variable also named `polyg`. So the name of a parameter is also the name of the variable that we can use in R for accesing the value of that parameter (thus, you should avoid using reserved R words as parameter names).

Spatial elements such as vector and raster layers are read using the `readOGR()` and `readGDAL()` commands (you do not have to worry about adding those commands to your description file, SEXTANTE will do it) and stored as `Spatial*DataFrame` objects. Table fields are stored as strings containing the name of the selected field.

Knowing that, we can now understand the first line of our example script (the first line not starting with a Python comment).

```
pts=spsample(polyg,numpoints,type="random")
```

The variable `polygon` already contains a `SpatialPolygonsDataFrame` object, so it can be usedto call the `spsample` method, just like the `numpoints` one, which indicates the number of points to add to the created sample grid.

Since we have declared an output of type vector named `out`, we have to create a variable named `out` and store a `Spatial*DataFrame` object in it (in this case, a `SpatialPointsDataFrame`). You can use any name for your intermediate variables. Just make sure that the variable storing your final result has the same name that you used to declare it, and contains a suitable value.

In this case, the result obtained from the `spsample` method has to be converted explicitly into a `SpatialPointsDataFrame` object, since it is itself an object of class `ppp`, which is not a suitable class to be retuned to SEXTANTE.

If you algorithm does not generate any layer, but a text result in the console instead, you have to tell SEXTANTE that you want the console to be shown once the execution is finished. To do so, just start the command lines that produce the results you want to print with the ">" sign. The output of all other lines will not be shown. For instance, here is the description file of an algorithms that performs a normality test on a given field (column) of the attributes of a vector layer:

```
##layer=vector
##field=field layer
##nortest=group
library(nortest)
>lillie.test(layer[[field]])
```

The output ot the last line is printed, but the output of the first is not (and neither are the otputs from other command lines added automatically by SEXTANTE).

If your algorithm creates any kind of graphics (using the `plot()` method), add the following line:

```
##showplots
```

This will cause SEXTANTE to redirect all R graphical outputs to a temporary file, which will be later opened once R execution has finished

Both graphics and console results will be shown in the SEXTANTE results manager.

For more information, please check the script files provided with SEXTANTE. Most of them are rather simple and will greatly help you understand how to create your own ones.

A note about libraries: `rgdal` and `maptools` libraries are loaded by default so you do not have to add the corresponding *library()* commands. However, other additional libraries that you might need have to be explicitly loaded. Just add the necessary commands at the beginning of your script. You also have to make sure that the corresponding packages are installed in the R distribution used by SEXTANTE.