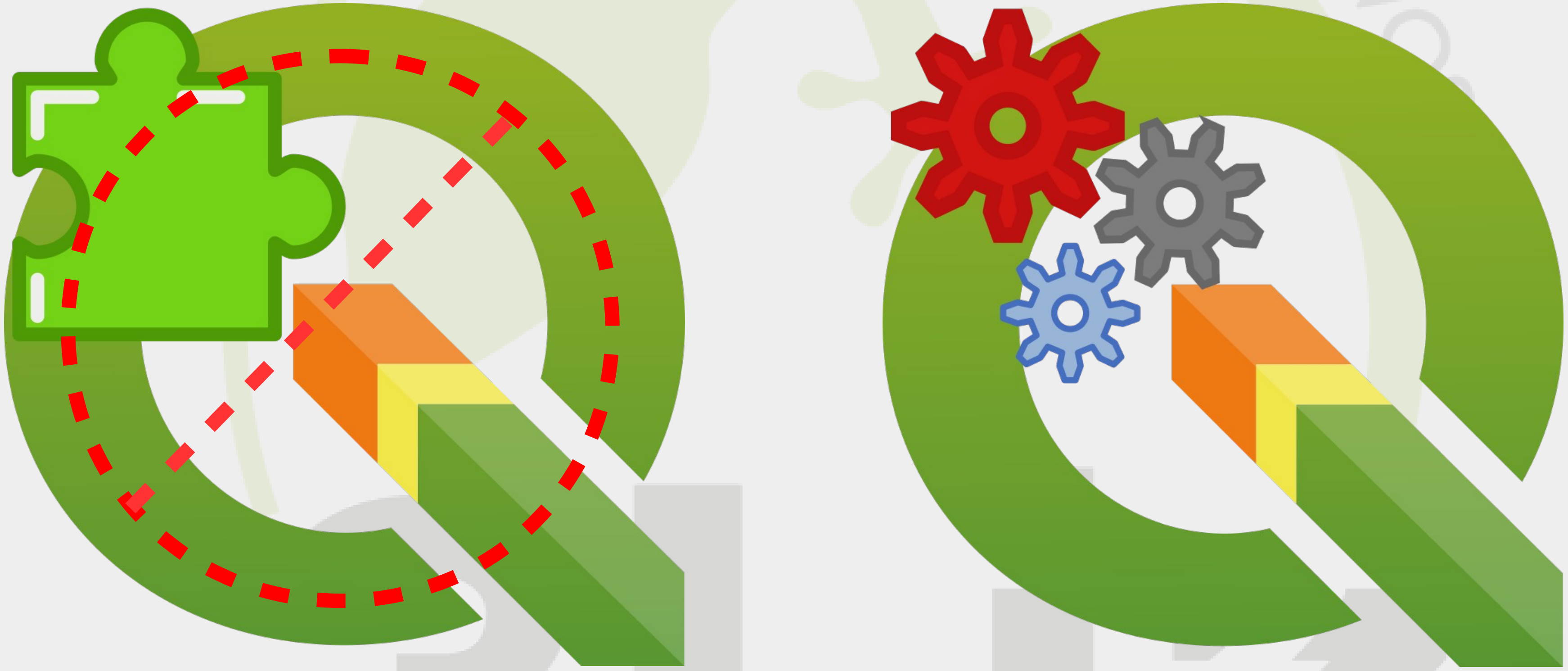
Plus besoin de plugins, seulement des algorithmes



3Liz SARL

Créé en mars 2007

Services QGIS, QGIS Server et Lizmap

Nous sommes 7 depuis septembre 2019

inspection visuel et analyse de réseaux d'eaux usés et pluviales

Ajout des plugins Python

Présenté au FOSS4G 2007 à Victoria

Publié le 20 october 2007

Le plugin Sextante

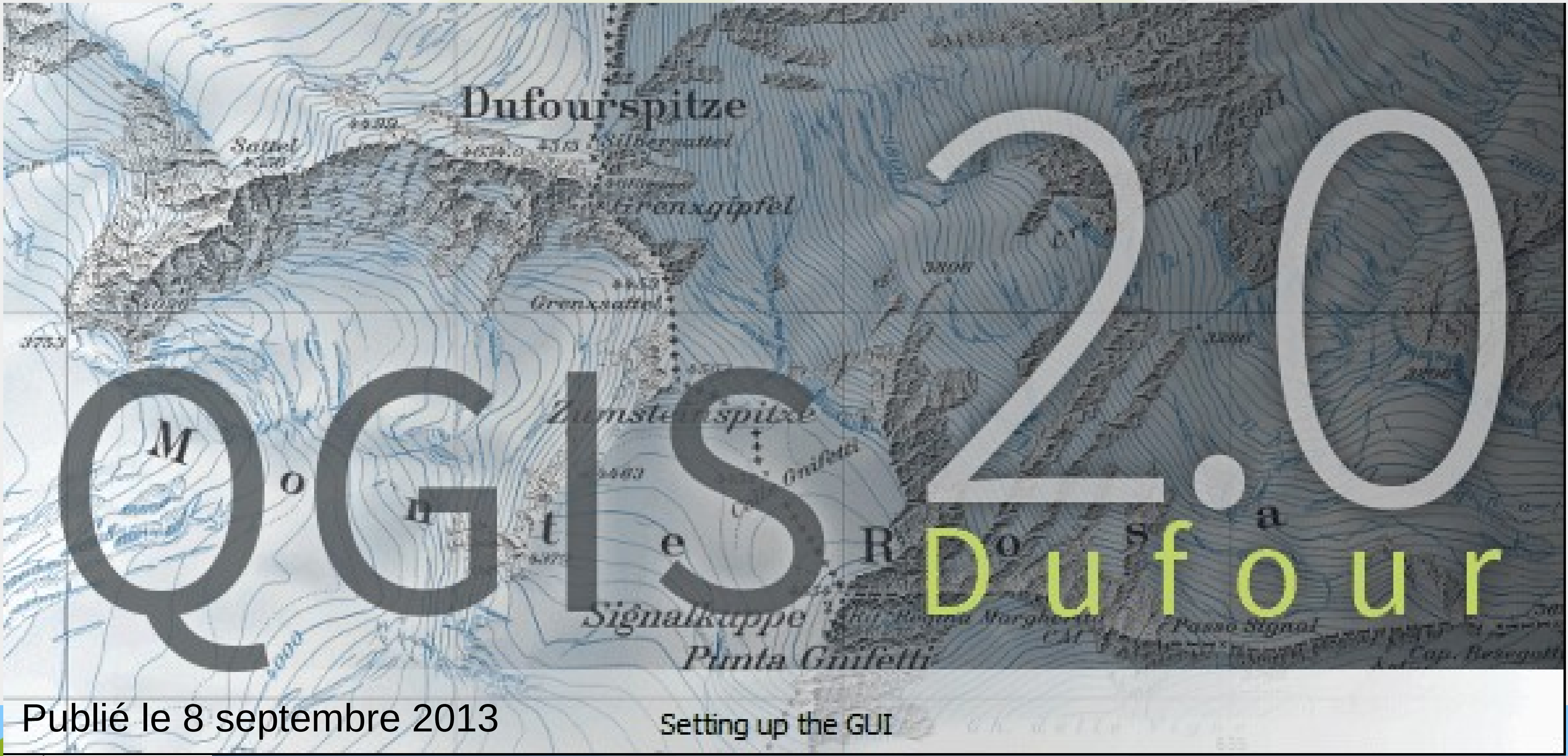
Le 21 mars 2012

« Just a quick comment that might be interesting related to that topic. I am about to release the first version of the SEXTANTE platform for QGIS. It contains a toolbox, a graphical modeler, script creator, ..., a batch processing interface, history, and much more. ... I wanted to wait a bit more until it is more or less stable, but since I see some action in the QGIS processing area, I think it is worth mentioning it now, so you can consider it. ... »

Le plugin Sextante

Publié le 21 mars 2012

De Sextante à Processing



Publié le 8 septembre 2013

W21: Spatial Analysis With QGIS And its Processing Module

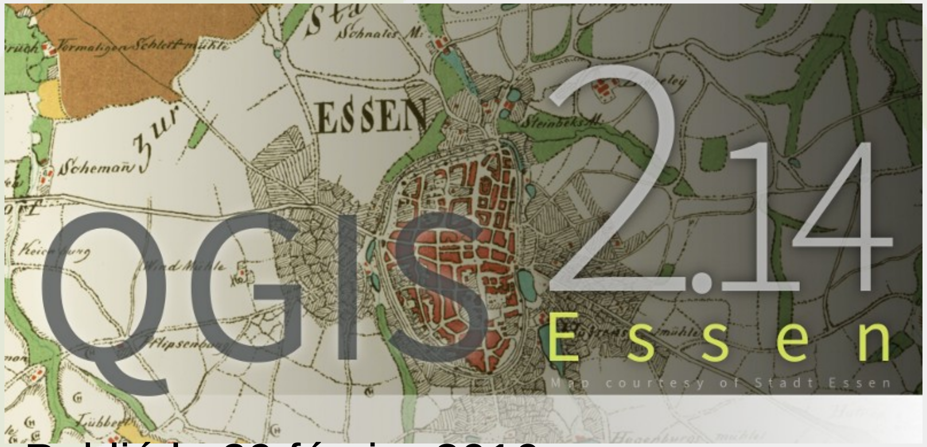
Conference International des utilisateurs QGIS 2016

- Processing is not an analysis framework
  - If you develop an analysis plugin, PLEASE, use Processing
  - Use processing for your plugins, even if they don't perform data analysis

Conference International des utilisateurs QGIS 2016

- Processing is not an analysis framework
  - Rethink how you write plugins
  - Allow your plugin functionality to be used like a library
  - And create Processing algorithms for methods in that library

Conference International des utilisateurs QGIS 2016



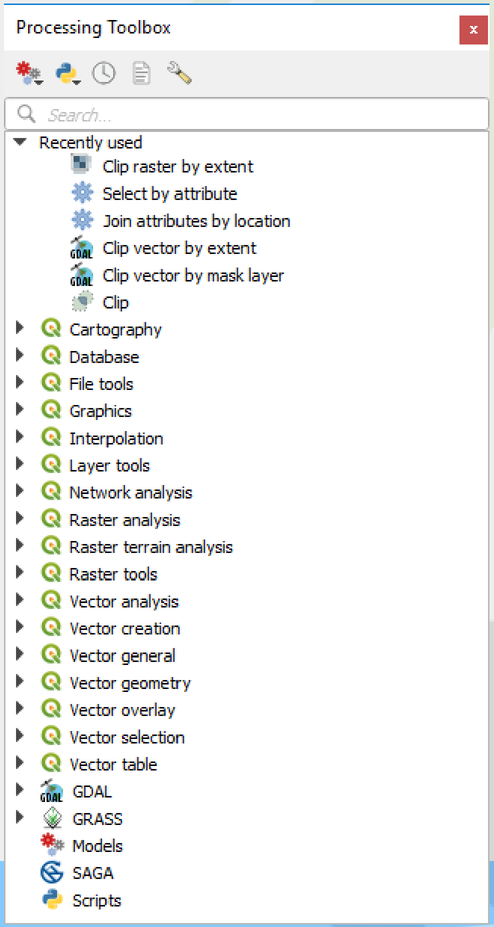
Publié le 29 février 2016

Publié le 21 octobre 2016

Processing dans le coeur QGIS

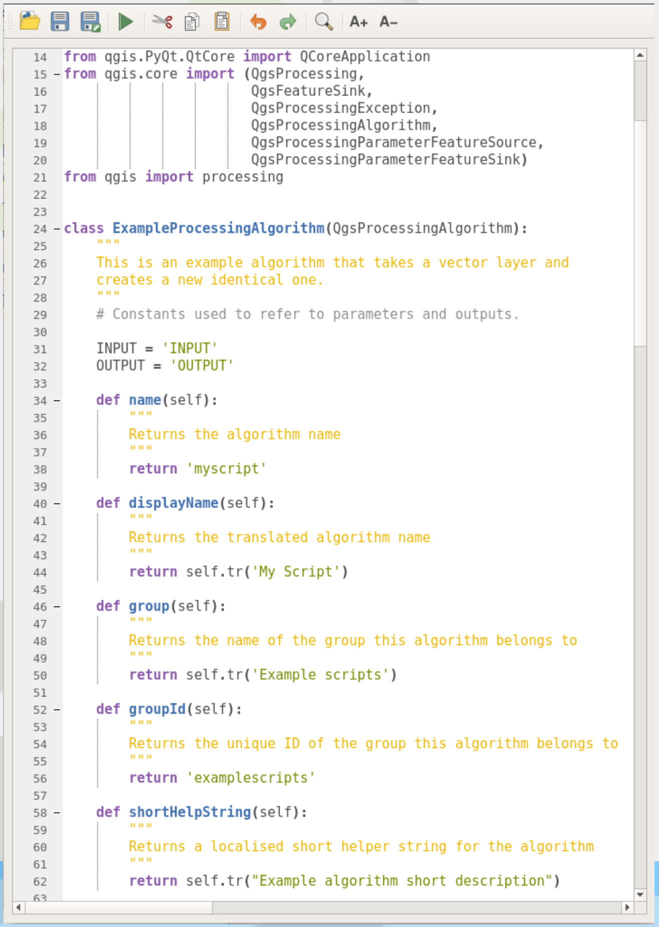
Publié le 23 février 2018

Nouveau module traitement


Processing Toolbox

Search...

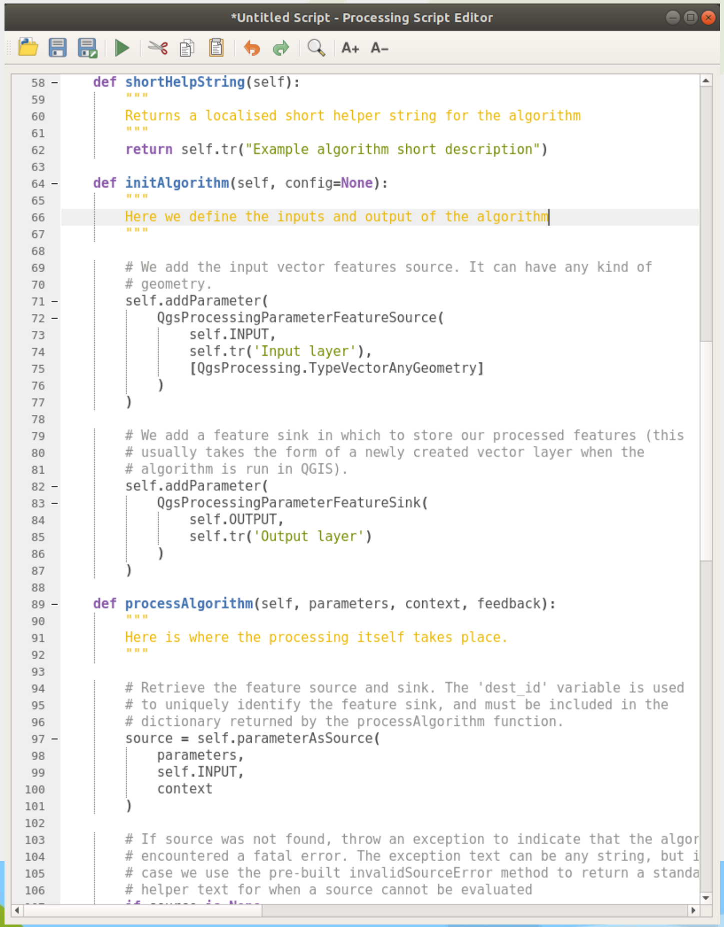
- Recently used
  - Clip raster by extent
  - Select by attribute
  - Join attributes by location
  - Clip vector by extent
  - Clip vector by mask layer
  - Clip
- Cartography
- Database
- File tools
- Graphics
- Interpolation
- Layer tools
- Network analysis
- Raster analysis
- Raster terrain analysis
- Raster tools
- Vector analysis
- Vector creation
- Vector general
- Vector geometry
- Vector overlay
- Vector selection
- Vector table
- GDAL
- GRASS
- Models
- SAGA
- Scripts



```python
from qgis.PyQt.QtCore import QCoreApplication
from qgis.core import (QgsProcessing,
                       QgsFeatureSink,
                       QgsProcessingException,
                       QgsProcessingAlgorithm,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterFeatureSink)
from qgis import processing


class ExampleProcessingAlgorithm(QgsProcessingAlgorithm):
    """
    This is an example algorithm that takes a vector layer and
    creates a new identical one.
    """

    # Constants used to refer to parameters and outputs.

    INPUT = 'INPUT'
    OUTPUT = 'OUTPUT'

    def name(self):
        """
        Returns the algorithm name
        """
        return 'myscript'

    def displayName(self):
        """
        Returns the translated algorithm name
        """
        return self.tr('My Script')

    def group(self):
        """
        Returns the name of the group this algorithm belongs to
        """
        return self.tr('Example scripts')

    def groupId(self):
        """
        Returns the unique ID of the group this algorithm belongs to
        """
        return 'examplescripts'

    def shortHelpString(self):
        """
        Returns a localised short helper string for the algorithm
        """
        return self.tr("Example algorithm short description")
```

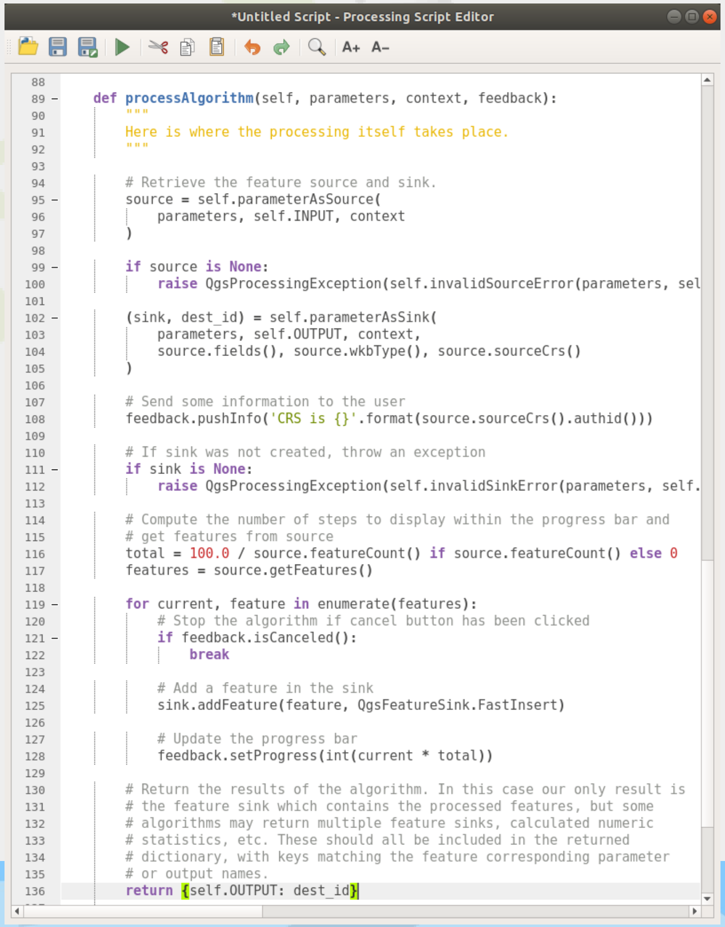
Plus besoin de plugins,
seulement des algorithmes

Nouveau module traitement



*Untitled Script - Processing Script Editor

```
    def shortHelpString(self):
        """
        Returns a localised short helper string for the algorithm
        """
        return self.tr("Example algorithm short description")

    def initAlgorithm(self, config=None):
        """
        Here we define the inputs and output of the algorithm
        """

        # We add the input vector features source. It can have any kind of
        # geometry.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
                self.INPUT,
                self.tr('Input layer'),
                [QgsProcessing.TypeVectorAnyGeometry]
            )
        )

        # We add a feature sink in which to store our processed features (this
        # usually takes the form of a newly created vector layer when the
        # algorithm is run in QGIS).
        self.addParameter(
            QgsProcessingParameterFeatureSink(
                self.OUTPUT,
                self.tr('Output layer')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        """
        Here is where the processing itself takes place.
        """

        # Retrieve the feature source and sink. The 'dest_id' variable is used
        # to uniquely identify the feature sink, and must be included in the
        # dictionary returned by the processAlgorithm function.
        source = self.parameterAsSource(
            parameters,
            self.INPUT,
            context
        )

        # If source was not found, throw an exception to indicate that the algor
        # encountered a fatal error. The exception text can be any string, but i
        # case we use the pre-built invalidSourceError method to return a standa
        # helper text for when a source cannot be evaluated
```

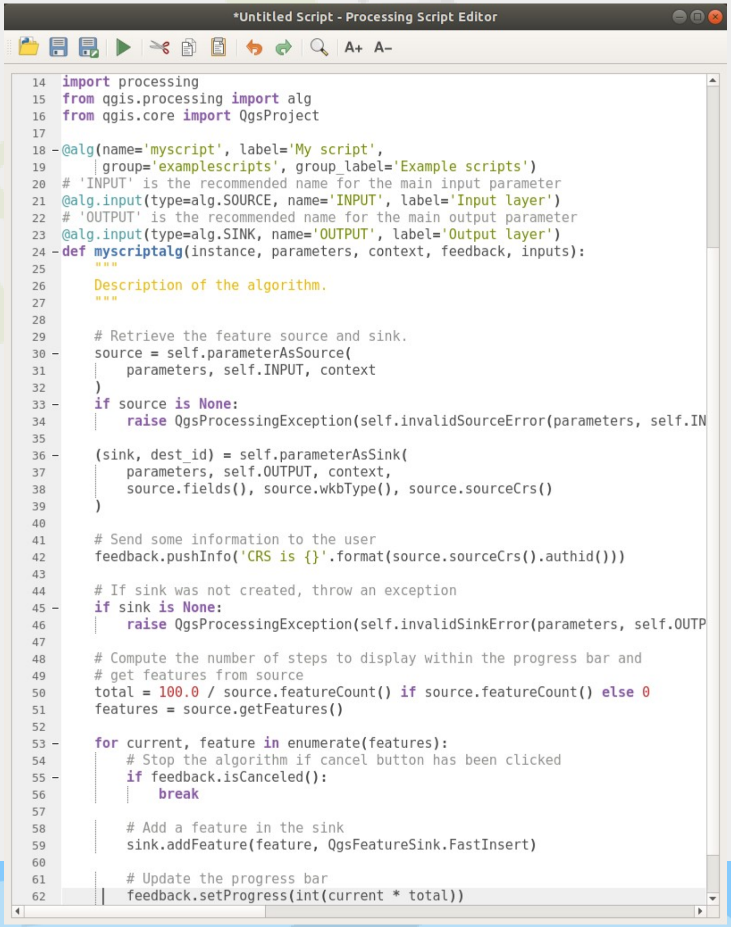


*Untitled Script - Processing Script Editor

```

    def processAlgorithm(self, parameters, context, feedback):
        """
        Here is where the processing itself takes place.
        """

        # Retrieve the feature source and sink.
        source = self.parameterAsSource(
            parameters, self.INPUT, context
        )

        if source is None:
            raise QgsProcessingException(self.invalidSourceError(parameters, sel

        (sink, dest_id) = self.parameterAsSink(
            parameters, self.OUTPUT, context,
            source.fields(), source.wkbType(), source.sourceCrs()
        )

        # Send some information to the user
        feedback.pushInfo('CRS is {}'.format(source.sourceCrs().authid()))

        # If sink was not created, throw an exception
        if sink is None:
            raise QgsProcessingException(self.invalidSinkError(parameters, self.

        # Compute the number of steps to display within the progress bar and
        # get features from source
        total = 100.0 / source.featureCount() if source.featureCount() else 0
        features = source.getFeatures()

        for current, feature in enumerate(features):
            # Stop the algorithm if cancel button has been clicked
            if feedback.isCanceled():
                break

            # Add a feature in the sink
            sink.addFeature(feature, QgsFeatureSink.FastInsert)

            # Update the progress bar
            feedback.setProgress(int(current * total))

        # Return the results of the algorithm. In this case our only result is
        # the feature sink which contains the processed features, but some
        # algorithms may return multiple feature sinks, calculated numeric
        # statistics, etc. These should all be included in the returned
        # dictionary, with keys matching the feature corresponding parameter
        # or output names.
        return {self.OUTPUT: dest_id}
```

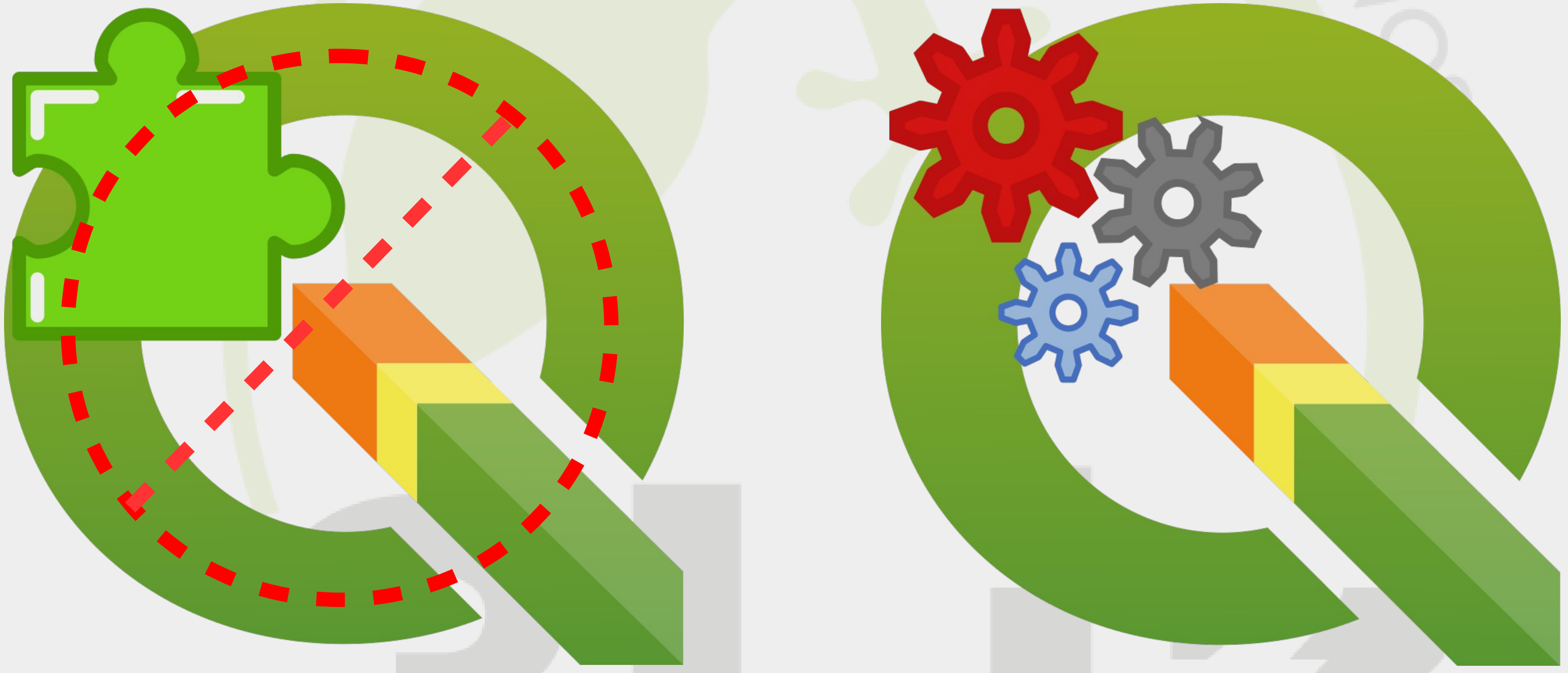
Script traitement simplifié

Publié le 22 février 2019



```
import processing
from qgis.processing import alg
from qgis.core import QgsProject

@alg(name='myscript', label='My script',
     group='examplescripts', group_label='Example scripts')
# 'INPUT' is the recommended name for the main input parameter
@alg.input(type=alg.SOURCE, name='INPUT', label='Input layer')
# 'OUTPUT' is the recommended name for the main output parameter
@alg.input(type=alg.SINK, name='OUTPUT', label='Output layer')
def myscriptalg(instance, parameters, context, feedback, inputs):
    """
    Description of the algorithm.
    """

    # Retrieve the feature source and sink.
    source = self.parameterAsSource(
        parameters, self.INPUT, context
    )
    if source is None:
        raise QgsProcessingException(self.invalidSourceError(parameters, self.IN

    (sink, dest_id) = self.parameterAsSink(
        parameters, self.OUTPUT, context,
        source.fields(), source.wkbType(), source.sourceCrs()
    )

    # Send some information to the user
    feedback.pushInfo('CRS is {}'.format(source.sourceCrs().authid()))

    # If sink was not created, throw an exception
    if sink is None:
        raise QgsProcessingException(self.invalidSinkError(parameters, self.OUTP

    # Compute the number of steps to display within the progress bar and
    # get features from source
    total = 100.0 / source.featureCount() if source.featureCount() else 0
    features = source.getFeatures()

    for current, feature in enumerate(features):
        # Stop the algorithm if cancel button has been clicked
        if feedback.isCanceled():
            break

        # Add a feature in the sink
        sink.addFeature(feature, QgsFeatureSink.FastInsert)

        # Update the progress bar
        feedback.setProgress(int(current * total))
```

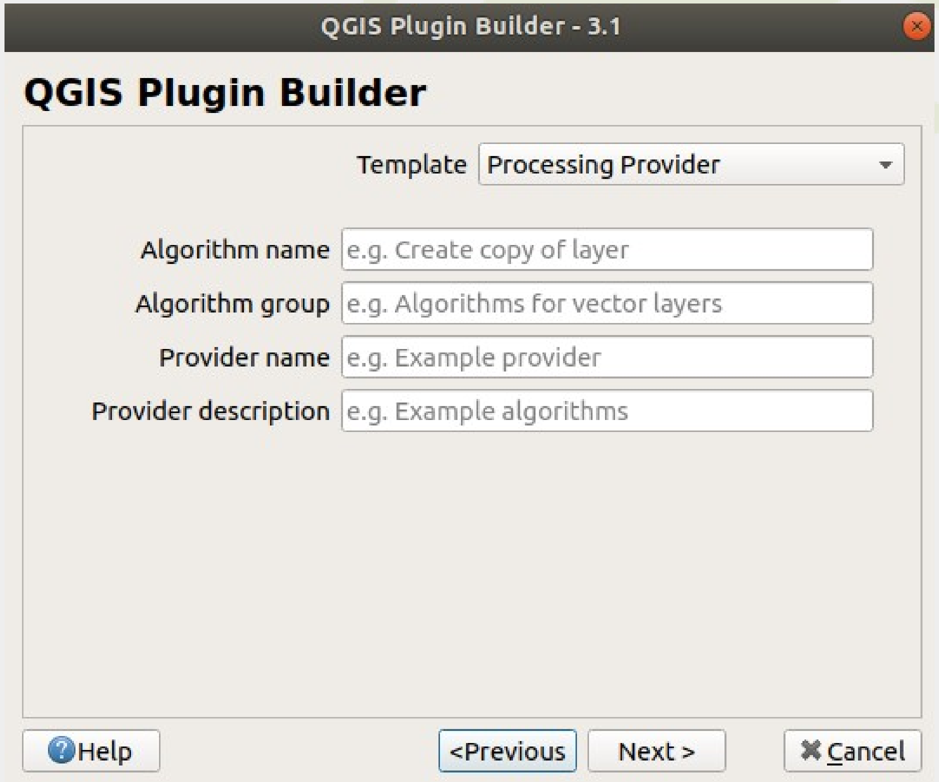
Plus besoin de plugins, seulement des algorithmes



Plugins are not dead

- L'interface utilisateur du module traitement !!!
- Ajouter un fournisseur d'algorithmes au module traitement
- Ajouter une interface utilisateur friendly

Plugins are not dead


QGIS Plugin Builder - 3.1

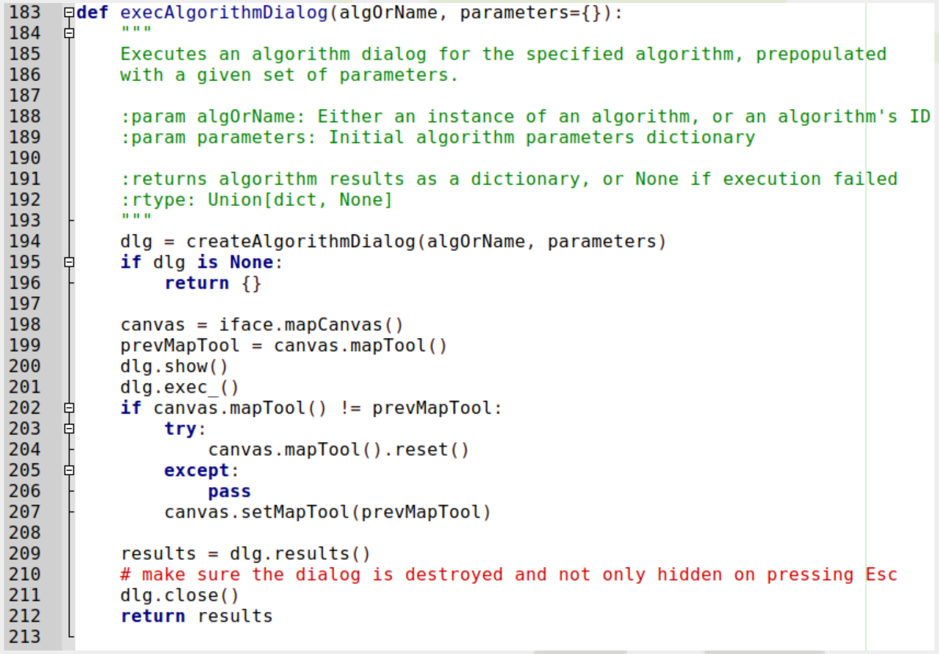
QGIS Plugin Builder

Template	Processing Provider
Algorithm name	e.g. Create copy of layer
Algorithm group	e.g. Algorithms for vector layers
Provider name	e.g. Example provider
Provider description	e.g. Example algorithms

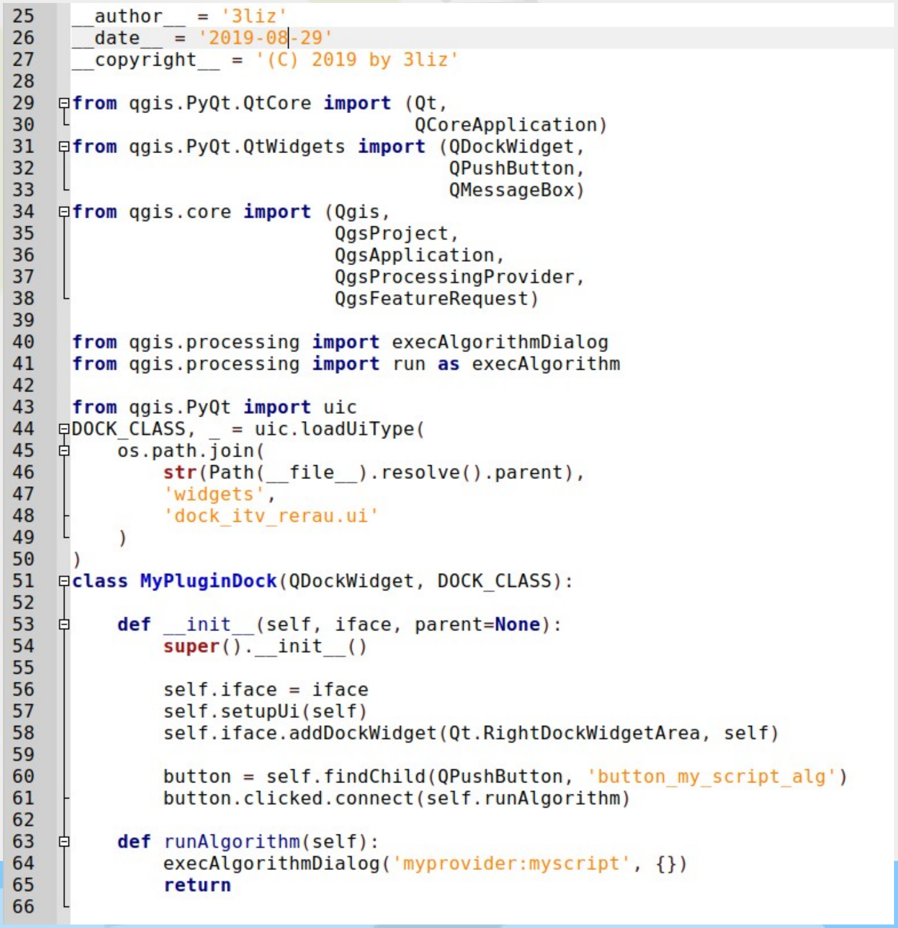
- Write Python algorithms
- Distribute your algorithms
- Add a dedicated User Interface or toolbar

Plugins are not dead

Ajouter une interface utilisateur dédié ou une barre à outils



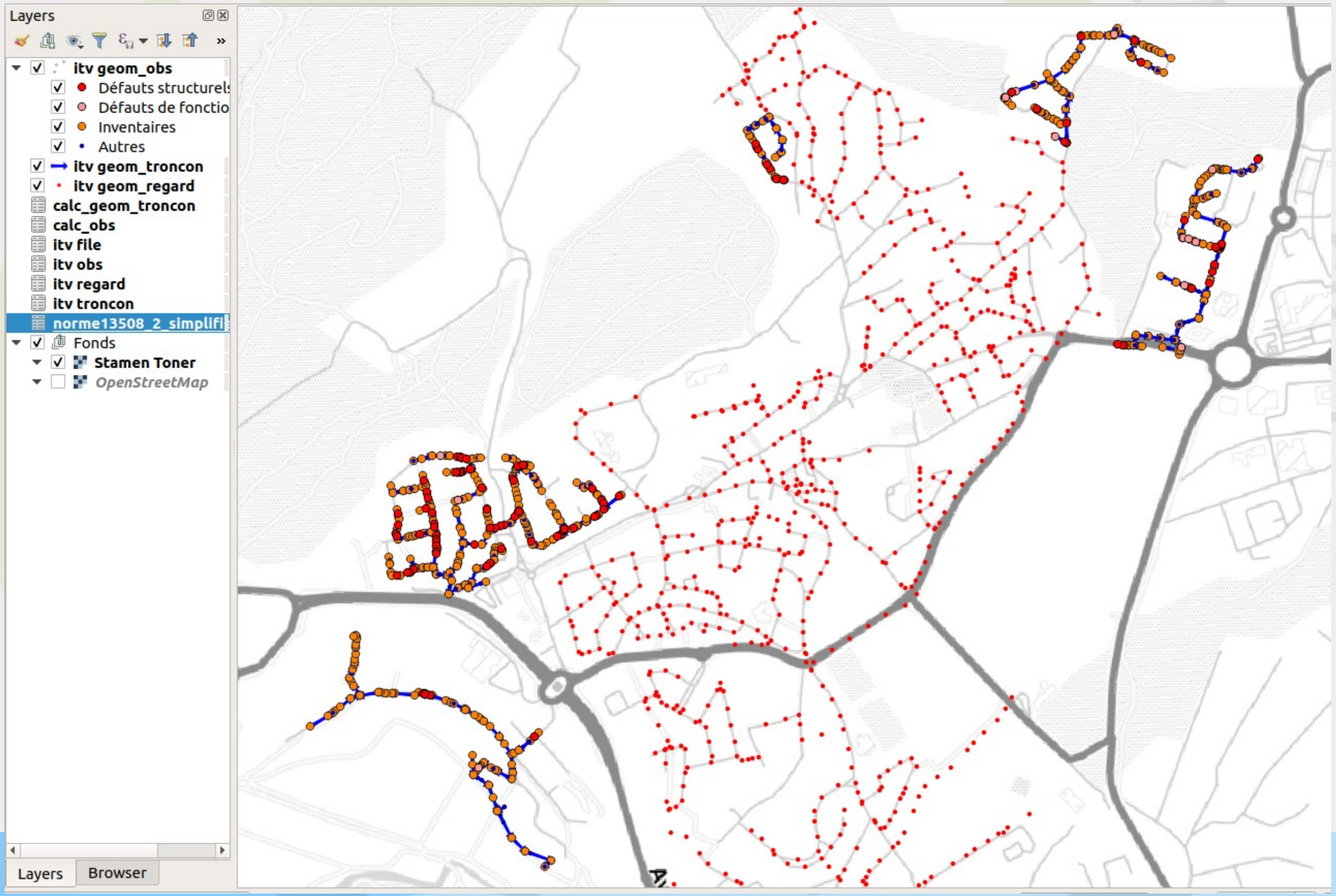
```python
def execAlgorithmDialog(algOrName, parameters={}):
    """
    Executes an algorithm dialog for the specified algorithm, prepopulated
    with a given set of parameters.

    :param algOrName: Either an instance of an algorithm, or an algorithm's ID
    :param parameters: Initial algorithm parameters dictionary

    :returns algorithm results as a dictionary, or None if execution failed
    :rtype: Union[dict, None]
    """
    dlg = createAlgorithmDialog(algOrName, parameters)
    if dlg is None:
        return {}

    canvas = iface.mapCanvas()
    prevMapTool = canvas.mapTool()
    dlg.show()
    dlg.exec_()
    if canvas.mapTool() != prevMapTool:
        try:
            canvas.mapTool().reset()
        except:
            pass
        canvas.setMapTool(prevMapTool)

    results = dlg.results()
    # make sure the dialog is destroyed and not only hidden on pressing Esc
    dlg.close()
    return results
```



```python
__author__ = '3liz'
__date__ = '2019-08-29'
__copyright__ = '(C) 2019 by 3liz'

from qgis.PyQt.QtCore import (Qt,
                              QCoreApplication)
from qgis.PyQt.QtWidgets import (QDockWidget,
                                 QPushButton,
                                 QMessageBox)
from qgis.core import (Qgis,
                       QgsProject,
                       QgsApplication,
                       QgsProcessingProvider,
                       QgsFeatureRequest)

from qgis.processing import execAlgorithmDialog
from qgis.processing import run as execAlgorithm

from qgis.PyQt import uic
DOCK_CLASS, _ = uic.loadUiType(
    os.path.join(
        str(Path(__file__).resolve().parent),
        'widgets',
        'dock_itv_rerau.ui'
    )
)
class MyPluginDock(QDockWidget, DOCK_CLASS):

    def __init__(self, iface, parent=None):
        super().__init__()

        self.iface = iface
        self.setupUi(self)
        self.iface.addDockWidget(Qt.RightDockWidgetArea, self)

        button = self.findChild(QPushButton, 'button_my_script_alg')
        button.clicked.connect(self.runAlgorithm)

    def runAlgorithm(self):
        execAlgorithmDialog('myprovider:myscript', {})
        return
```

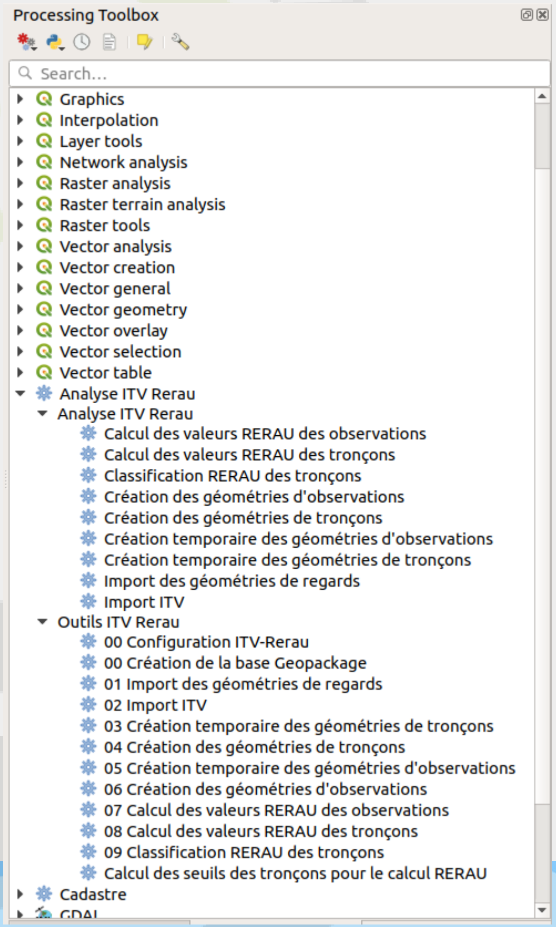
Exemple : inspection visuel et analyse

Exemple : inspection visuel et analyse



Exemple : inspection visuel et analyse

- Besoins:
  - Créer des packages ou schéma
  - Configurer le projet QGIS
  - Importer des fichiers
  - Géolocaliser les obs
  - Lancer des analyses



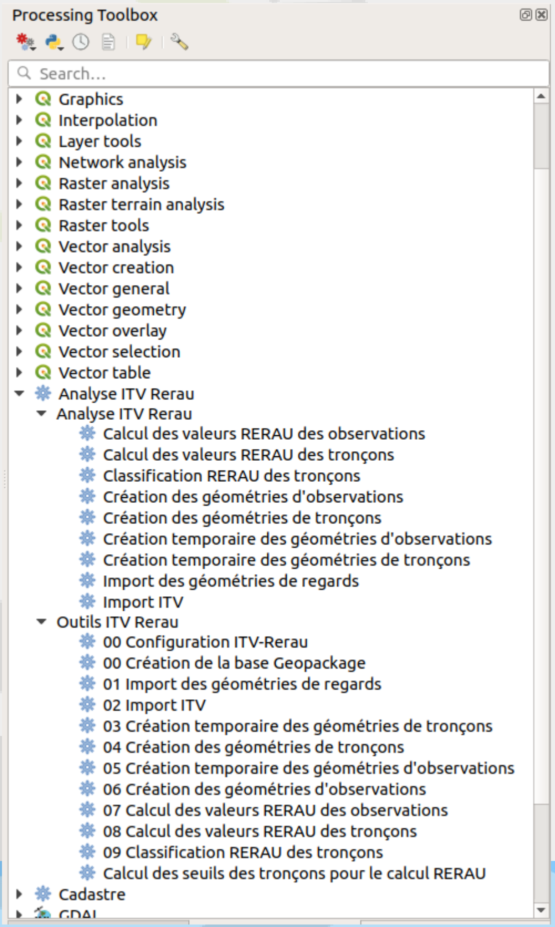
Processing Toolbox

Search...

- Graphics
- Interpolation
- Layer tools
- Network analysis
- Raster analysis
- Raster terrain analysis
- Raster tools
- Vector analysis
- Vector creation
- Vector general
- Vector geometry
- Vector overlay
- Vector selection
- Vector table
- Analyse ITV Rerau
  - Analyse ITV Rerau
    - Calcul des valeurs RERAU des observations
    - Calcul des valeurs RERAU des tronçons
    - Classification RERAU des tronçons
    - Création des géométries d'observations
    - Création des géométries de tronçons
    - Création temporaire des géométries d'observations
    - Création temporaire des géométries de tronçons
    - Import des géométries de regards
    - Import ITV
  - Outils ITV Rerau
    - 00 Configuration ITV-Rerau
    - 00 Création de la base Geopackage
    - 01 Import des géométries de regards
    - 02 Import ITV
    - 03 Création temporaire des géométries de tronçons
    - 04 Création des géométries de tronçons
    - 05 Création temporaire des géométries d'observations
    - 06 Création des géométries d'observations
    - 07 Calcul des valeurs RERAU des observations
    - 08 Calcul des valeurs RERAU des tronçons
    - 09 Classification RERAU des tronçons
    - Calcul des seuils des tronçons pour le calcul RERAU
- Cadastre
- GDAL

Exemple : inspection visuel et analyse

- Limites:
  - Trop d’entrées par défault
    - Utilisation des variables de projet
    - Réutilisationdes algos dans les algos
  - Pas d’interface dédié

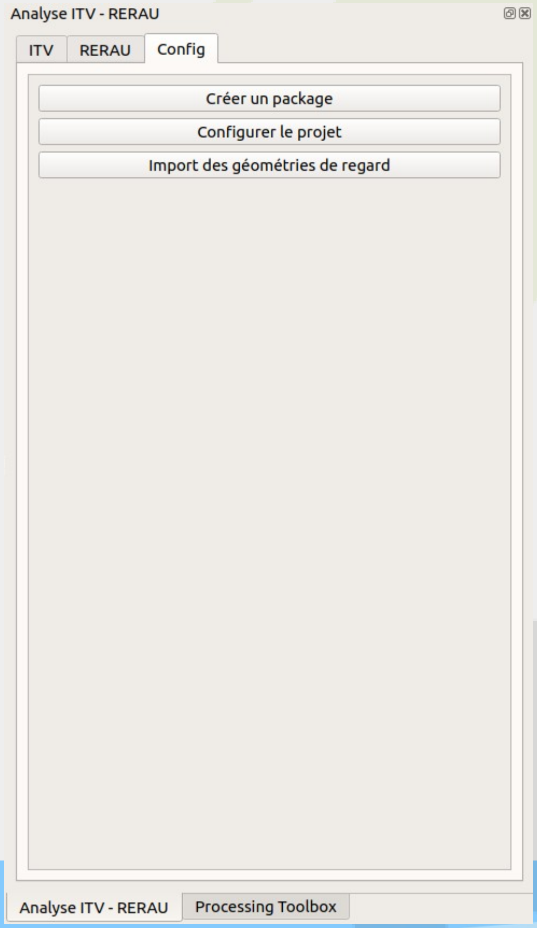


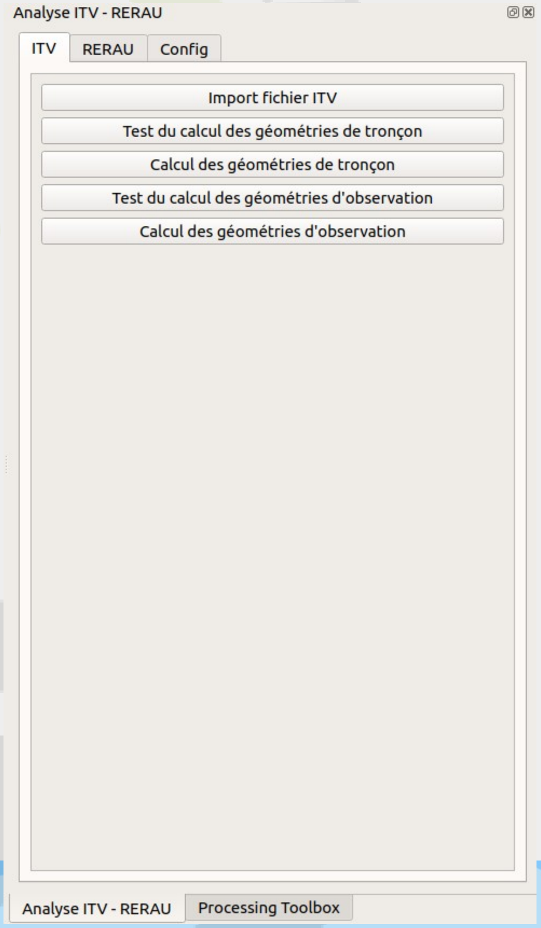
Processing Toolbox

Search...

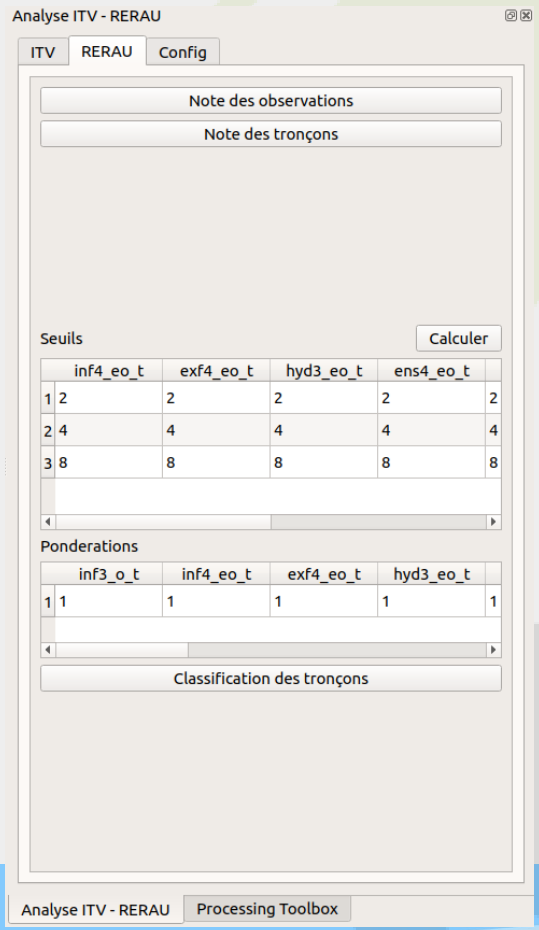
- Graphics
- Interpolation
- Layer tools
- Network analysis
- Raster analysis
- Raster terrain analysis
- Raster tools
- Vector analysis
- Vector creation
- Vector general
- Vector geometry
- Vector overlay
- Vector selection
- Vector table
- Analyse ITV Rerau
  - Analyse ITV Rerau
    - Calcul des valeurs RERAU des observations
    - Calcul des valeurs RERAU des tronçons
    - Classification RERAU des tronçons
    - Création des géométries d'observations
    - Création des géométries de tronçons
    - Création temporaire des géométries d'observations
    - Création temporaire des géométries de tronçons
    - Import des géométries de regards
    - Import ITV
  - Outils ITV Rerau
    - 00 Configuration ITV-Rerau
    - 00 Création de la base Geopackage
    - 01 Import des géométries de regards
    - 02 Import ITV
    - 03 Création temporaire des géométries de tronçons
    - 04 Création des géométries de tronçons
    - 05 Création temporaire des géométries d'observations
    - 06 Création des géométries d'observations
    - 07 Calcul des valeurs RERAU des observations
    - 08 Calcul des valeurs RERAU des tronçons
    - 09 Classification RERAU des tronçons
    - Calcul des seuils des tronçons pour le calcul RERAU
- Cadastre
- GDAL

Exemple : inspection visuel et analyse




Analyse ITV - RERAU
ITV
RERAU
Config
Import fichier ITV
Test du calcul des géométries de tronçon
Calcul des géométries de tronçon
Test du calcul des géométries d'observation
Calcul des géométries d'observation
Analyse ITV - RERAU
Processing Toolbox

Exemple : inspection visuel et analyse



Analyse ITV - RERAU

ITV | RERAU | Config

Note des observations

Note des tronçons

Seuils

Calculer

	inf4_eo_t	exf4_eo_t	hyd3_eo_t	ens4_eo_t	
1	2	2	2	2	2
2	4	4	4	4	4
3	8	8	8	8	8

Ponderations

	inf3_o_t	inf4_eo_t	exf4_eo_t	hyd3_eo_t	
1	1	1	1	1	1

Classification des tronçons

Analyse ITV - RERAU | Processing Toolbox

- Pours:
  - Réduire le temps de développement
  - Re-utiliser des algorithmes
- Contres:
  - Le module traitement n'est pas complet

Avantages du module traitement

- Fenêtres d’algorithme
- Modèles
- Batch
- WPS

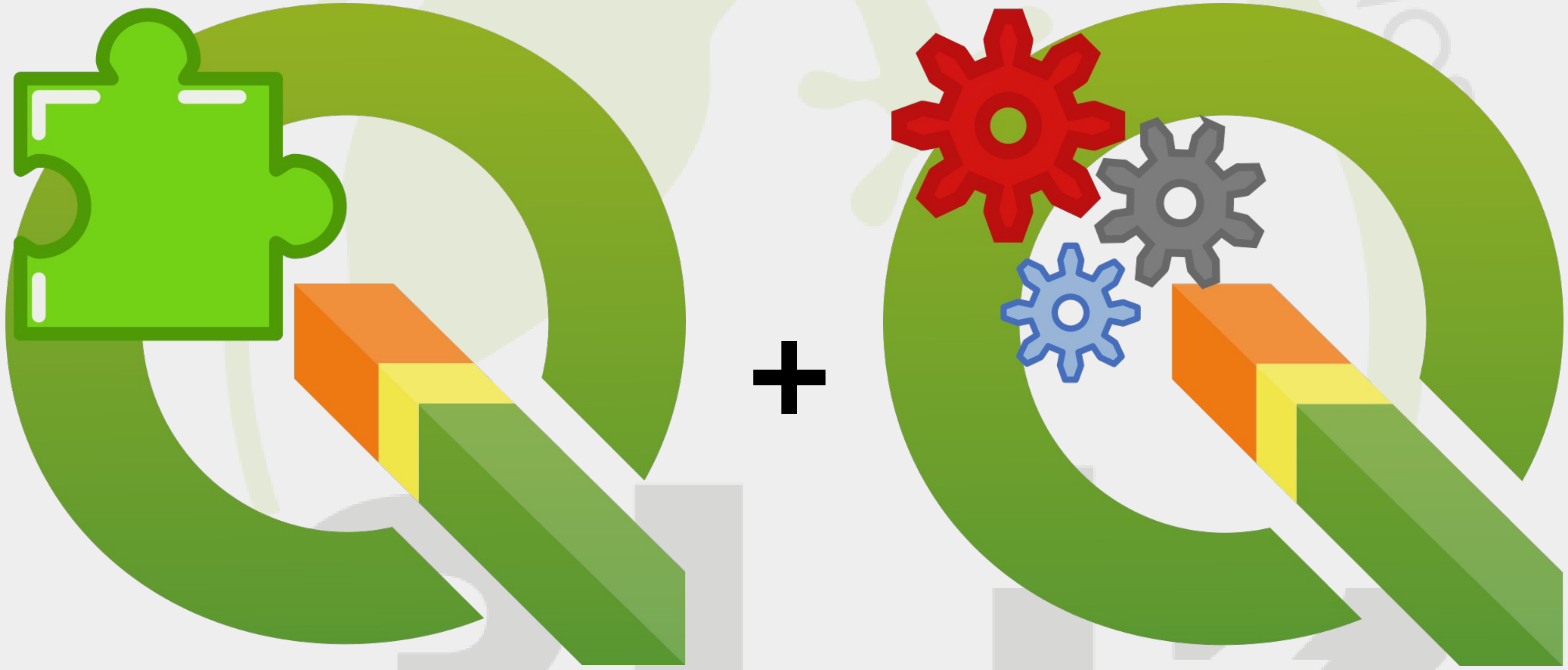
Avantages du module traitement

- Drain Sewer Visual Inspection
- RAEPA
- QuickOSM ?
- Cadastre ?
- Urbanisme ?

https://github.com/3liz

LIBEREZ VOS SIG !

Conclusion



Plus besoin de plugins, seulement des algorithmes

Merci !
Questions ?