RAPPORT DE STAGE CHEZ HALLER WASSER + PARTNER SA

CERTIFICAT COMPLÉMENTAIRE EN GÉOMATIQUE

FACULTÉ DES SCIENCES ET SCIENCES DE LA SOCIÉTÉ – UNIGE

VALENTINA BUFFON

JUIN-OCTOBRE 2020

Notice analytique	
Auteur :	Valentina Buffon
Titre:	Mise au point d'outils géomatiques sur diverses plateformes
Résumé:	Le certificat complémentaire en géomatique proposé par la faculté des Sciences et des Sciences de la Société à l'UNIGE permet aux étudiants d'acquérir des notions en géomatique et d'utiliser divers logiciels SIG. La participation à un stage est une étape fondamentale pour l'obtention de la certification requise. Le stage effectué en entreprise, dans le bureau d'ingénieurs géomètres Haller Wasser + partner SA, m'a permis de mettre en pratique les notions acquises durant la formation, mais aussi de compléter mes connaissances, notamment par l'utilisation quotidienne de logiciels SIG (QGIS, ArcMap, MapInfo), de FME, un logiciel pour la transformation de jeux de données, et de Qt Designer, permettant de créer des interfaces graphiques. J'ai également pu pratiquer divers types de langages (HTML, JavaScript, VBScript et Python) pour effectuer toutes sortes de paramétrages. Le sujet principal du stage a été la mise en place d'un SIG complet sur QGIS pour les données professionnelles du bureau, ainsi que la formation des utilisateurs à ce nouveau logiciel, dans l'optique de remplacer MapInfo. Des couches du SITG (shapefiles et connexion au MapServer) et des couches propres au bureau (fichiers texte, shapefiles et fichiers tab) ont été intégrées au projet. Une barre d'outils personnalisée a été ajoutée, contenant toutes sortes de fonctionnalités utiles au quotidien. De plus, dans les propriétés des couches, certains paramètres ont été ajoutés, notamment en créant des formulaires, des infobulles, des actions, et en personnalisant des symbologies et des étiquettes. Des mises en pages prédéfinies (standard ou complexes) ont été ajoutées au projet, via le gestionnaire de mises en page et en tant que modèles. Finalement, un plugin pour l'export d'une sélection de points au format CSV a été développé et ajouté à la barre d'outils. Des projets annexes ont été réalisés sur QGIS, dans l'optique de créer des cartes thématiques afin de mieux comprendre la problématique. Sur ArcMap, il a été question de créer des mises en pages prédéfinies, afin de les charger par le biais de la barre d'outils TopoGeo. Des fichiers Layer Files ont été créés pour diverses couches, contenant des symboles et des étiquettes personnalisés, en fonction des attentes du bureau. Ces fichiers lyr ont été intégrés dans un fichier xml, afin de charger les différentes symbologies créées. D'autres projets annexes ont été effectués sur ArcMap, en utilisant des outils dans le ArcToolbox, et en créant des mises en pages. Finalement, l'ETL spatial FME a permis de réaliser des développements élaborés et organisés, soit pour intégrer des données dans le projet principal sur QGIS, soit pour modifier et améliorer des données du bureau. Des traitements ont été effectués entre autres sur des bases de données (GDB ou MDB), des shapefiles,

	des raster (GeoTIFF), des nuages de points Lidar (LAS), et des fichiers AutoCAD (DWG). Il a été question de créer diverses grilles de points (séparés de 100 mètres, puis de 2 mètres) à partir des modèles numériques, afin de les charger dans QGIS et d'afficher les valeurs d'élévation de ces points. En parallèle, un développement a permis de récupérer les données du MNS, grâce aux grilles de points créées sur le MNT et le MNA. Un développement annexe a permis de géo-référencer des dossiers ou devis qui n'avaient pas les bonnes coordonnées GPS, grâce à des bases de données du bureau et à des couches du SITG (adresses, parcelles et parcelles historiques) qui sont géo-référencées.
Summary:	The complementary certificate in geomatics offered by the Faculty of Science and Society Sciences at UNIGE enables students to acquire notions in geomatics and to use various GIS software. The participation in an internship is a fundamental step in obtaining the required certification. The internship carried out in a company, at the Haller Wasser + partner SA surveying engineering office, allowed me to put into practice the notions acquired during the training, but also to complete my knowledge, in particular through the daily use of GIS software (QGIS, ArcMap, MapInfo), of FME, a software for the transformation of data sets, and of Qt Designer, which allows the creation of graphic interfaces. I have also been able to practice various types of languages (HTML, JavaScript, VBScript and Python) to make all kinds of settings. The main subject of the internship was the implementation of a complete GIS on QGIS for professional office data, as well as the training of users to this new software, with a view to replacing MapInfo. Layers from the SITG (shapefiles and connection to MapServer) and layers specific to the office (text files, shapefiles and tab files) were integrated into the project. A customized toolbar has been added, containing all kinds of functionalities useful in daily use. In addition, in the layer properties, some parameters have been added, such as creating forms, tooltips, actions, and customizing symbologies and labels. Predefined layouts (standard or complex) have been added to the project, via the Layout Manager and as templates. Finally, a plugin for exporting a selection of points in CSV format was developed and added to the toolbar. Ancillary projects have been carried out on QGIS, with a view to creating thematic maps in order to better understand the issue. On ArcMap, predefined layouts were created in order to load them via the TopoGeo toolbar. Layer Files were created for various layers, containing custom symbols and labels, according to the expectations of the office. These lyr files have been integrated into an xml file, in order to load the different symbologies created. Other side projects were carried out on ArcMap, using tools in the ArcToolbox, and creating layouts. Finally, the FME spatial ETL was used to elaborate and organize developments, either to integrate data into the main project on QGIS, or to modify and improve office data. Processing were performed on databases (GDB or MDB), shapefiles,

	raster (GeoTIFF), Lidar point clouds (LAS), and AutoCAD (DWG) files, among others. The idea was to create various grids of points (separated by 100 meters and then by 2 meters) from the digital models, in order to load them into QGIS and display the elevation values of these points. At the same time, a development made it possible to retrieve the DSM data, thanks to the point grids created on the DTM and DEM. An ancillary development was aimed to geo-reference affairs or quotations that did not have the right GPS coordinates, thanks to office databases and layers from the SITG (addresses, parcels and historical parcels) that are geo-referenced.
Directeur:	M. Andrea de Bono
Jury:	M. Gregory Giuliani, M. Andrea de Bono, M. Gaëtan Martin
Date de l'examen:	2020-12-17
Diffusion autorisée:	Oui
Note de l'examen:	6
Volée:	Janvier-Juin 2020
Collation:	Nombre de pages : 123 Nombre de figures : 142 Nombre de tableaux : --
Sujet:	Rapport de stage pour la formation complémentaire en géomatique

Contenu

Introduction

Lors de mon stage de quatre mois et demi chez Haller Wasser, j'ai eu l'occasion d'utiliser des logiciels SIG tels que QGis, MapInfo et ArcMap, ainsi que FME, AutoCAD et Qt Designer. J'ai pu utiliser divers langages de programmation, notamment le HTML, JavaScript, VBScript et Python, via divers paramétrages.

Le projet principal du stage est réalisé sur QGis, dans l'optique de créer un projet propre au bureau, qui remplacerait le logiciel MapInfo. Ce projet QGis vise à fournir un outil pratique et rapide au personnel du bureau, soit pour la préparation avant terrain, soit pour faire toutes sortes de recherches en interne. L'interface est volontairement similaire à celle du SITG, pour que les utilisateurs trouvent leurs repères facilement. Le fonctionnement de QGis et le projet ont été présentés à tous les collaborateurs, pour qu'ils se familiarisent peu à peu à cette plateforme qui leur sera utile au quotidien.

En parallèle, d'autres projets annexes sont réalisés sur QGis, ArcMap et FME. Les développements réalisés sur FME ont pour finalité soit de transformer des données afin de les intégrer dans le projet QGis, soit de modifier et d'améliorer des données du bureau. L'utilisation d'ArcMap a essentiellement été dirigée dans l'optique de créer des symbologies personnalisées, afin de les charger dans n'importe quel projet par le biais de la barre d'outils TopoGeo, un ESRI Add-in.

Toutes les étapes de mise en place des divers projets sur les plateformes susmentionnées seront présentées dans ce rapport, avec des illustrations et des exemples.

Présentation du bureau Haller Wasser + partner SA

Le bureau Haller Wasser + partner SA est né de la fusion entre deux bureaux d'ingénieurs géomètres officiels genevois, Christian Haller et Frédéric Wasser. Ce bureau comporte une quarantaine d'employés, contenant un large panel de profils : géomètres brevetés, géomètres, géomaticiens, informaticiens, architectes, aides de terrain et apprentis. Il traite divers domaines du territoire et de l'environnement bâti, et possède de multiples compétences, passant par l'étude de projet, la phase de chantier, la mensuration officielle, la stratégie foncière, l'expertise, la géomatique et le Building Information Modeling (BIM), ce dernier étant un outil permettant la modélisation des données par l'utilisation de la 3D.

Les projets réalisés par le bureau sont nombreux, comportant des travaux menés à petite échelle, notamment dans le domaine de la mensuration officielle, en réalisant des divisions ou réunions parcellaires, des travaux de cadastration, la constitution de servitudes etc. D'autres projets de plus grande envergure sont accomplis ou en cours, comme le projet de remaniement routier de la Jonction au Grand-Saconnex (JAG), en collaboration avec l'Office Fédéral des Routes (OFROU), ou bien le projet des Communaux d'Ambilly, visant à urbaniser un secteur à Thônex (projet MICA). Par ailleurs, le bureau possède du matériel de pointe ; théodolites, niveaux, lasers scanners, drones, bathymètres, inclinomètres et lasers, pour réaliser toutes sortes de travaux sur le terrain.

Le bureau Haller Wasser + partner SA m'a donné l'opportunité de faire mon stage de géomatique dans ses locaux, me permettant de mettre en pratique les notions acquises durant les cinq mois de formation à l'Université de Genève.

1. Projet principal : remplacement de MapInfo par QGis

QGIS est un logiciel SIG (système d'information géographique) open-source et multiplateforme, publié sous licence GPL (General Public Licence). Ce programme a connu une récente évolution ; sa constante mise à jour et la multitude de développeurs y travaillant laissent présager un développement pérenne du logiciel (https://www.eode.ch/2016/05/06/quest-ce-que-qgis-et-pourquoi-se-former/). Les développeurs du logiciel s'efforcent de donner une interface agréable et compréhensible, et sont à l'écoute des éventuelles demandes des utilisateurs. Par ailleurs, il existe une communauté d'utilisateurs, échangeant leur avis et leurs connaissances sur des plateformes, notamment sur GeoRezo et gis.stackexchange. Ces forums se révèlent très précieux lorsque l'on ne trouve pas de réponses ou de solutions dans la documentation QGis. Néanmoins, cette documentation est à jour et présente divers aspects et paramètres du logiciel, qui sont utiles lorsque l'on débute sur QGis et que l'on souhaite en comprendre les fondements.

MapInfo est également un logiciel SIG, utilisé pour la cartographie et l'analyse géographique. Ce logiciel permet la réalisation de cartes numériques, la superposition de couches, la personnalisation de l'interface etc. Le bureau Haller Wasser n'a pas voulu renouveler l'expérience avec MapInfo ; la version qu'ils avaient devenait obsolète, et ils ne souhaitaient pas racheter une licence. QGis les a séduits par sa facilité d'utilisation, sa vaste palette d'outils, sa renommée et sa gratuité, qui est un avantage non négligeable, par rapport à d'autres logiciels SIG similaires payants. Par ailleurs, l'aspect personnalisable de ce programme est un avantage lorsque l'on souhaite créer un SIG complet et uniforme, propre à un bureau, pour loger tous les utilisateurs à la même enseigne.

1.1 But du projet

Ce projet sur QGis a pour objectif de remplacer le logiciel MapInfo, qui a été utilisé par le bureau durant de nombreuses années. Par ce chapitre, nous allons expliquer et illustrer les différents outils nécessaires au quotidien sur QGis, ainsi que toutes les méthodes permettant de réaliser les étapes de construction du projet.

Trois barres d'outils personnalisées ont été créées, afin d'avoir tous les outils nécessaires à portée de main. La barre d'outils « Outils HW » contient divers boutons utiles pour toute sorte d'utilisation, la barre d'outils « Mise en page » contient les outils nécessaires pour la sélection d'entités et l'impression d'une mise en page type, et finalement la barre d'outils « Ajout points » permet de créer des nouvelles entités géoréférencées, à un shapefile.

Outils HW : 

Mise en page : 

Ajout points : 

1.2 Couches

Le projet compte diverses couches provenant du domaine de la mensuration officielle, téléchargées sur le SITG au format shapefile. Il contient également des couches propres au bureau Haller Wasser, dont certaines ont été reprises de MapInfo et sont au format .tab. Elles ont par conséquent été exportées en .shp pour garder une cohérence avec les autres couches. D'autres couches du bureau sont au format texte, et ont été rajoutées par le biais d'un ajout de couche de texte délimité. Finalement, certaines couches ont été intégrées par le biais d'une connexion au MapServer du SITG.

1.3 Marche à suivre

Dans un premier temps, nous allons présenter les fonctionnalités de ces barres d'outils, comme elles ont été présentées aux employés du bureau Haller Wasser. Puis, dans la partie « méthodes », nous allons entrer dans les détails et expliquer comment ces fonctionnalités ont été implémentées.

1.4 Présentation du projet

1.4.1 Barres d'outils

Les barres d'outils ont été intégrées au projet grâce au plugin « Customize Toolbars »  , téléchargé dans les extensions. Ce plugin permet de déplacer les outils « Qgis tools » qui nous intéressent dans la boîte « Outils HW », qui est le nom de la boîte à outil personnalisée (Figure 1). Il est possible de créer autant de barres d'outils que nécessaire, par le biais du bouton « New ToolBar ». La création de barres d'outils s'avère très précieuse lorsque l'on souhaite créer un projet pour plusieurs utilisateurs ; en effet, cela permet de les aiguiller et les diriger dans le programme. L'interface de QGis peut contenir beaucoup de boutons, ce qui peut rendre la tâche difficile à des utilisateurs novices. Pour des utilisateurs plus avérés, cela est également pratique pour ranger les outils en fonction de leur utilisation. Il est également possible de ne pas afficher les barres d'outils par défaut, et de garder que les outils nécessaires à l'utilisateur, en les glissant dans une barre d'outils personnalisée.

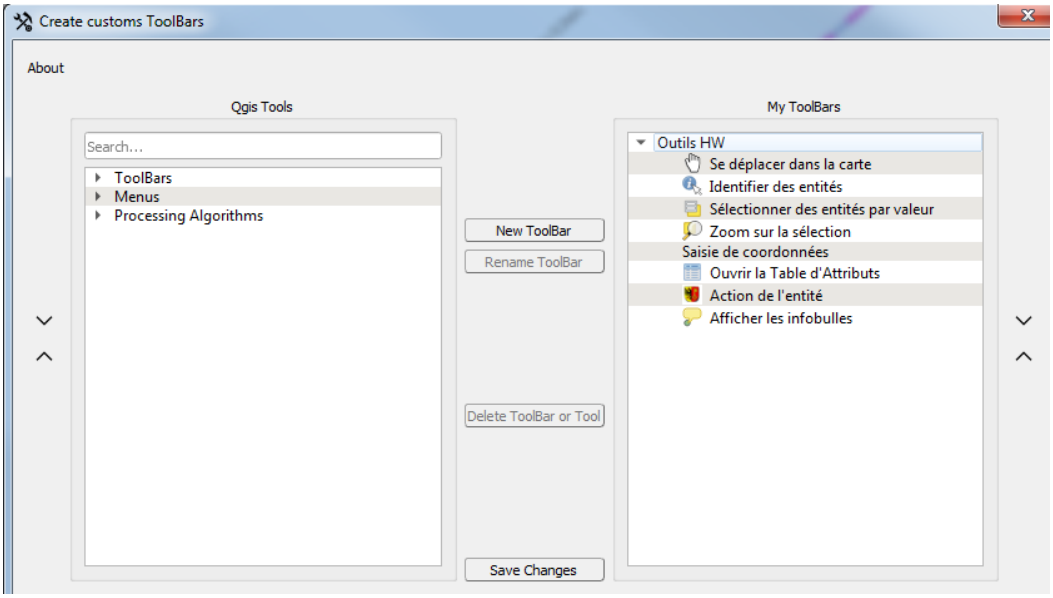


Figure 1: Fenêtre de boîte à outils personnalisable dans le plugin « Customize Toolbar ».

Voici la liste des trois barres avec les outils qu'elles contiennent :

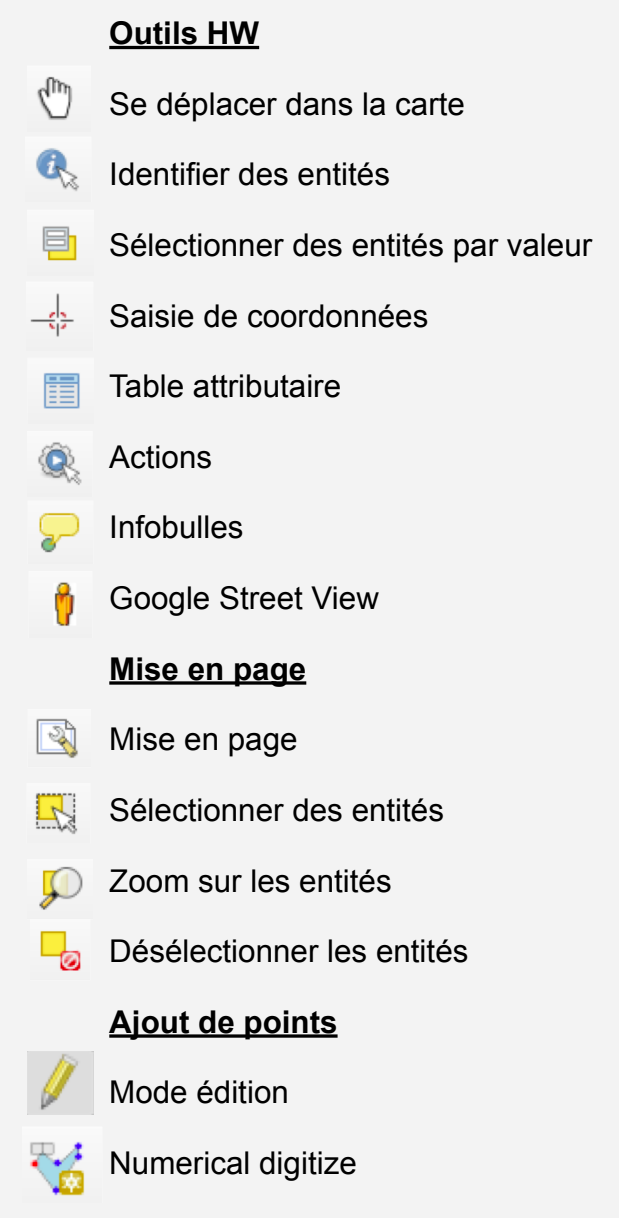


Figure 2 : Liste des trois barres d'outils créées sur QGis.

1.4.2 Fonctionnalités des outils

Outils « HW »

1a. Identifier des entités

Cet outil permet d'interroger la couche sélectionnée dans l'arborescence des couches (Figure 3). Pour que l'information apparaisse, il faut sélectionner une entité de la couche en question sur la carte. *Exemple* : cliquer sur une parcelle provenant de la couche CAD_PARCELLE_MENSU, afin d'obtenir toutes les informations complémentaires à cette parcelle (Figure 4).

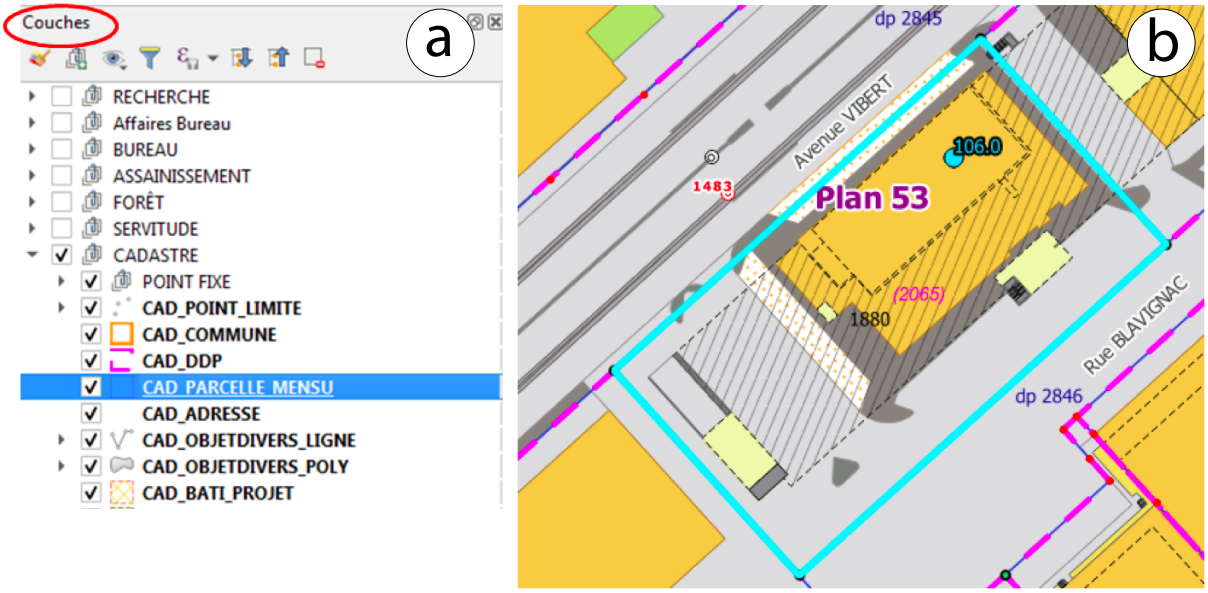


Figure 3 : **a)** Sélection d'une couche dans l'arborescence et **b)** sélection d'une entité (ici parcelle), via l'outil « i ».

Résultats de l'identification

Entité	Valeur
CAD_PARCELLE_MENSU	
Titre	1880
(Dérivé)	
(Actions)	
EGRID	CH296583676326
COMMUNE	Carouge
NO_COMM	8
NO_PARCELL	1880
IDEDDP	8:1880
SURFACE	4890
MUTMAI	821968
MUTORI	821968
DATEDT	2019-07-18
PROVENANCE	terrestre
VALIDITE	en vigueur
TYPE_PROPR	privé
PLAN_RF	53
SHAPE_AREA	4890.28064889000
SHAPE_LEN	291.68439911400

Figure 4 : Résultat de l'identification de la parcelle sélectionnée dans la figure 3, contenant toutes les informations de la table attributaire pour cette parcelle (informations fournies par le SITG).

Pour se faciliter la tâche, il est préférable d'utiliser le clic droit de la souris lorsque l'on sélectionne un point sur la carte. L'outil va interroger toutes les couches qui sont superposées sur ce point, il n'est donc pas nécessaire de présélectionner une couche dans l'arborescence, mais il faut que les couches qui nous intéressent soient cochées. Les couches traversées par le clic vont apparaître, et il est possible de choisir celle qui nous intéresse (en cliquant sur la couche) ou bien de toutes les identifier

(identifier tout) (Figure 5). Les informations apparaissent à nouveau dans les résultats de l'identification, comme en figure 4.

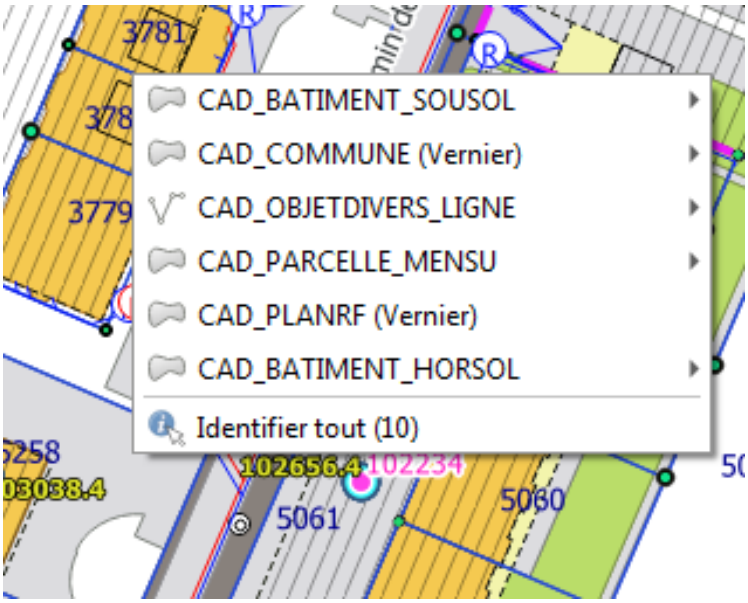


Figure 5 : Clic droit sur un point de la carte avec affichage de toutes les couches traversées par le clic.

1b. Identifier des entités sur un raster

Il faut sélectionner une couche raster dans l'arborescence (*par exemple* : MNA, MNT ou MNS) sur laquelle on souhaite identifier une entité, puis cliquer sur le point désiré sur la carte, et un résultat d'identification s'affichera sur la droite de la carte. Afin d'afficher le pixel sélectionné sur la carte (petit carré de sélection bleu ciel), il suffit de cliquer sur *pixel value* ou *stretched value* (Figure 6). Ici, le champ *pixel value* correspond à la valeur réelle du pixel, il indique donc l'altitude de l'élément sélectionné, car nous sommes sur le modèle numérique de hauteur (MNA). Le MNA correspond à la hauteur réelle de l'élément, obtenu grâce à la soustraction du modèle numérique de surface (MNS) par le modèle numérique de terrain (MNT). Et le champ *streched value* indique comment les valeurs des pixels sont étirées le long d'une rampe de couleurs (www.e-education.psu.edu).

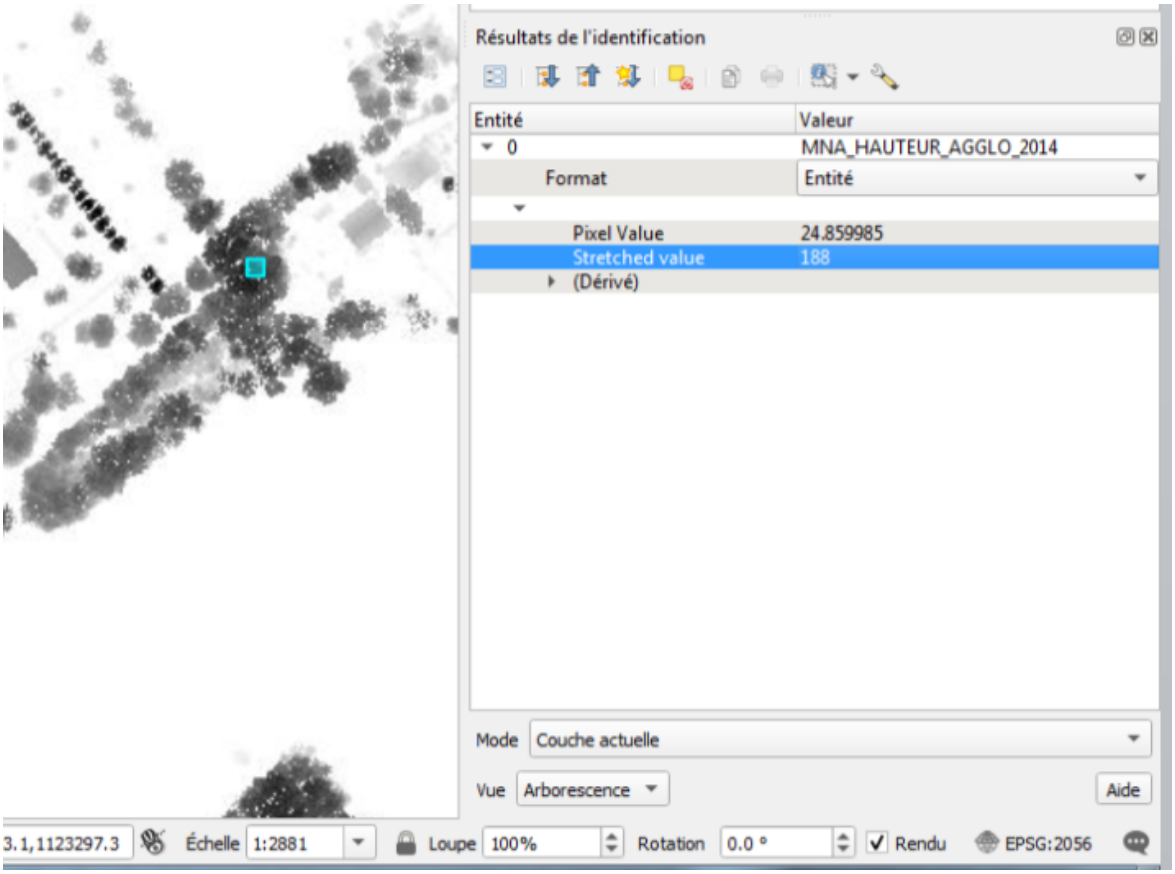


Figure 6 : Identification d'un raster, avec le point sélectionné mis en évidence sur la carte.

2. Recherche

Ce bouton permet de faire une recherche par attributs de la couche. La recherche se fait directement dans l'arborescence des couches, sous l'onglet « RECHERCHE » (Figure 7). Les couches de ce groupe sont des duplicatas de couches préexistantes, auxquels on a retiré la symbologie, pour ne pas avoir de doublons. Cela permet de placer ces couches tout en haut de l'arborescence, sous le même groupe, pour effectuer une recherche plus directe et rapide.

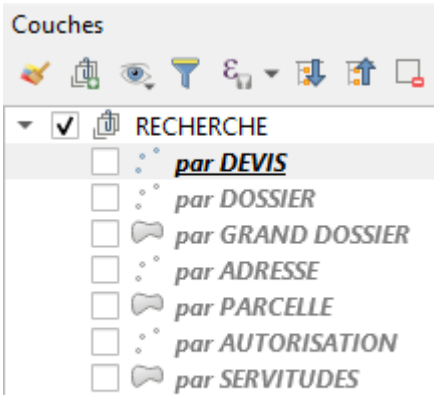


Figure 7 : Groupe de couches RECHERCHE contenant les duplicatas des couches des devis, dossiers, grands dossiers, adresses, parcelles, autorisations et servitudes.

Chaque couche contient quelques attributs utiles pour la recherche, qui ont été prédéfinis dans les propriétés de la couche, sous l'onglet « Formulaire d'attributs ». Plusieurs attributs sont affichés pour laisser le choix à l'utilisateur, mais pas toutes les cases ne doivent être remplies.

Afin d'effectuer une recherche, il faut sélectionner la couche désirée (*par exemple* : « par ADRESSE ») , puis cliquer sur le bouton « sélectionner des entités par valeur » dans la barre d'outils  .

Exemple de recherche par adresse

Les attributs choisis pour cette couche sont « adresse », « commune » et « code postal » (**Erreur ! Nous n'avons pas trouvé la source du renvoi.**). Une fois qu'une ou plusieurs de ces informations sont rentrées, cliquer sur :

- **Zoom sur les entités** : ce bouton n'est pas réellement un zoom mais un déplacement de la carte sur la zone en question. Il se peut que ça ne soit pas très précis et que l'entité n'apparaisse pas clairement. Après avoir cliqué sur le zoom, cliquer sur le clignotement des entités.

- **Clignotement des entités** : l'entité souhaitée va clignoter en rouge, ce qui est utile pour la repérer et éventuellement zoomer à nouveau dessus grâce à la molette de la souris.

- **Sélection des entités** : va sélectionner les entités souhaitées, mais ne va pas les faire apparaître sur la carte. Pour qu'un zoom soit opéré sur les entités sélectionnées, il faut cliquer sur l'icône « zoom sur la sélection »  .

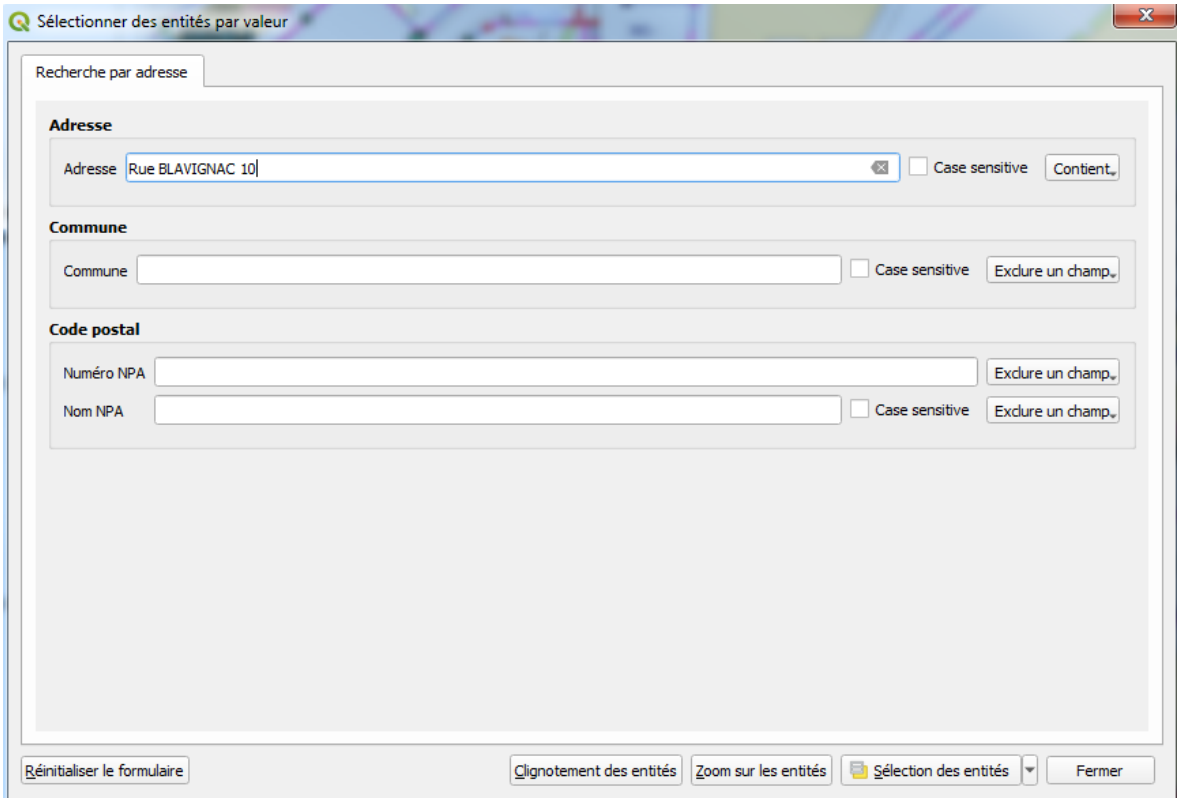


Figure 8 : Exemple de recherche par adresse, en utilisant l'outil « sélectionner des entités par valeur ».

3. Saisie de coordonnées

Cet outil permet de déterminer les coordonnées géographiques (X, Y) en chaque point. Il suffit de laisser défiler le curseur sur la carte et ensuite de cliquer sur un endroit pour obtenir les coordonnées. Cet outil est nécessaire lorsque l'on souhaite renseigner les coordonnées d'un nouveau dossier ou devis dans la base de données. Les coordonnées apparaissent en degrés décimaux (DD) et en latitude longitude, dans le système de coordonnées Suisse MN95.

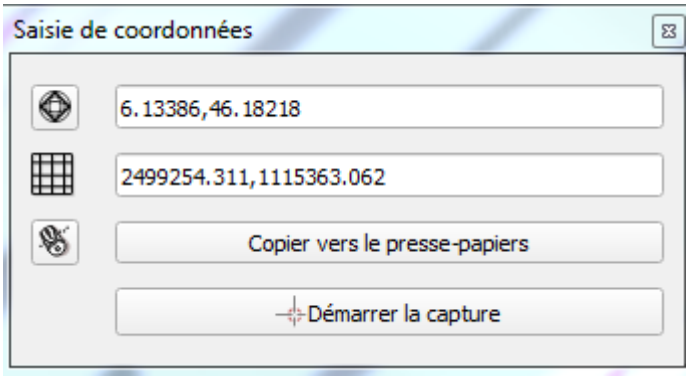


Figure 9 : Utilisation de l'outil « saisie de coordonnées ».

4. Table attributaire

Permet d'afficher la table attributaire de la couche sélectionnée dans l'arborescence. Cela peut s'avérer utile lorsque l'on souhaite trier des champs par ordre alphabétique ou numérique, ou lorsque l'on veut zoomer sur une entité connue, ou si l'on souhaite ne montrer que les entités sélectionnées.

5. Infobulles

L'infobulle s'affiche sur la carte lorsque l'on a sélectionné une couche dans l'arborescence, que l'on active le bouton « afficher les infobulles » et que l'on passe avec le curseur de la souris sur le point d'intérêt (*par exemple* : le point fixe 844, entouré en rouge dans la figure 10). L'infobulle va afficher une fenêtre contenant des informations de la table attributaire qui ont été prédéfinies. La création d'une infobulle se fait dans les propriétés de la couche, en codant au format HTML. Cette fonction a été implémentée uniquement aux couches des parcelles, des DDP, des bâtiments hors-sol et sous-sol, des points limites et des points fixes (PP, TRI et NIV PFA1-2-3).

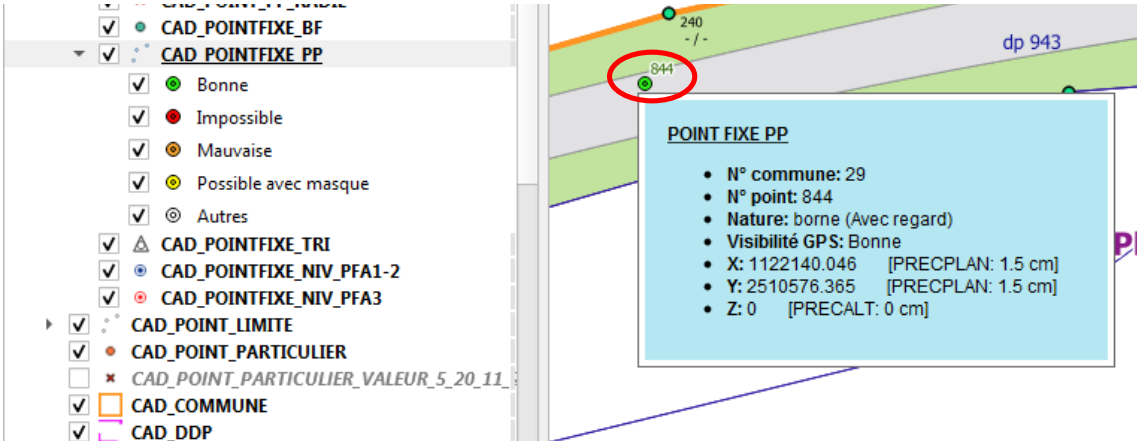


Figure 10 : Exemple d'affichage d'infobulle avec les données relatives au point fixe sur la carte.

6a. Actions

La création d'une action se fait dans les propriétés de la couche, en ajoutant le type d'action souhaité. Cela va permettre de définir une action propre à une couche, comme par exemple assigner un lien à une couche, ouvrir un fichier ou charger un shapefile sur l'endroit cliqué.

Les différentes actions implémentées sont listées ci-après :

-  Ouvrir la page web de l'extrait foncier lorsque l'on clique sur une **parcelle** ou un **DDP**. Ceci affichera les informations relatives à la parcelle ou au DDP sélectionné.
-  Ouvrir la page web du SAD (suivi des dossiers) lorsque l'on clique sur une **parcelle.**
-  Ouvrir la page web du SITG sur le PDF du **regard** sélectionné.
-  Ouvrir la page web du SITG avec la thématique du cadastre ou de l'aménagement lorsque l'on clique sur une **parcelle**.
-  Ouvrir la page web de SwissTopo sur les orthophotos 2020 à 10 cm (toute la Suisse) lorsque l'on clique sur une **parcelle**.
-  Ouvrir le PDF de la documentation QGis lorsque l'on clique sur une **parcelle**.
-  Ouvrir Google Maps lorsque l'on clique sur une **adresse**.
-  Ouvrir Google Itinéraire lorsque l'on clique sur une **adresse**, qui va ouvrir une page web avec l'itinéraire de l'adresse cliquée et l'adresse du bureau (rue Blavignac 10).

-  Charger l'orthophoto de l'endroit cliqué sur la couche **Index_Ortho2019.**
-  Charger la grille du modèle numérique de l'endroit cliqué sur les couches **Index_MNT_2019**, **Index_MNS_2019** ou **Index_MNA_2019**.

Afin de lancer une action, il y a deux méthodes :

1. Il faut d'abord sélectionner (dans l'arborescence), la couche sur laquelle l'action a été implémentée. Puis appuyer sur la flèche à droite du bouton d'action  , sélectionner l'action désirée et cliquer à l'endroit souhaité sur la carte. **NB** : il faut cliquer à l'intérieur d'une parcelle si l'action est implémentée sur une parcelle, de même qu'il faut cliquer à l'intérieur d'un DDP ou sur le numéro de l'adresse.

2. Utiliser l'outil d'identification des entités (i), et faire un clic droit sur un point de la carte. Se placer avec la souris sur la couche désirée, et les actions relatives à cette couche apparaitront (Figure 11).

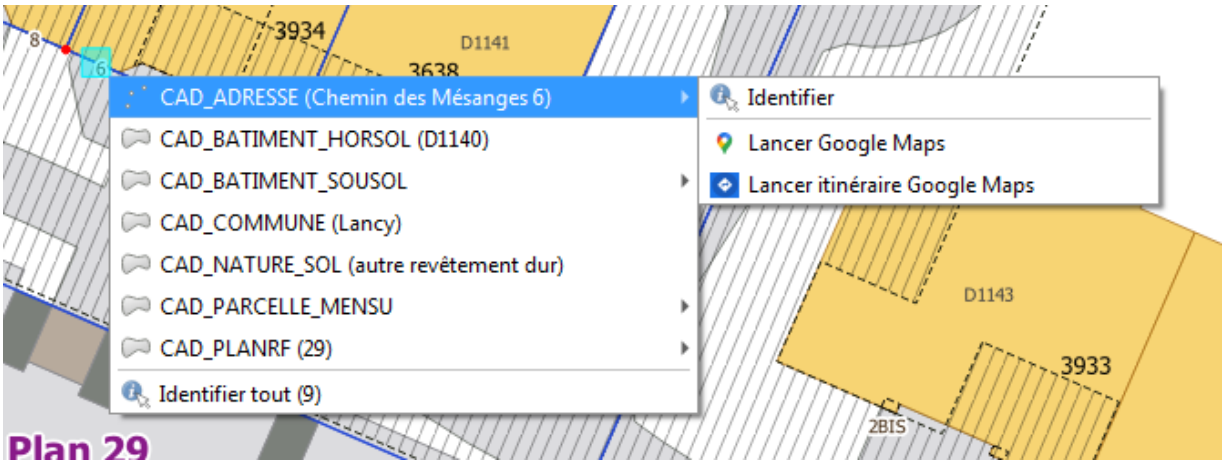


Figure 11 : Affichage des actions par le biais de l'outil d'identification d'entités.

6b. Action pour charger la dalle orthophoto de l'endroit cliqué

Les orthophotos ne sont pas chargées dans l'arborescence des couches car elles ralentissent le projet. Par conséquent, une action a été implémentée sur la couche « Index_Ortho2019 », qui se trouve sous le groupe « RECHERCHE », au sommet de l'arborescence. Cette couche est une grille au format shapefile, qui est un contour des différentes tuiles des orthophotos (Figure 12).

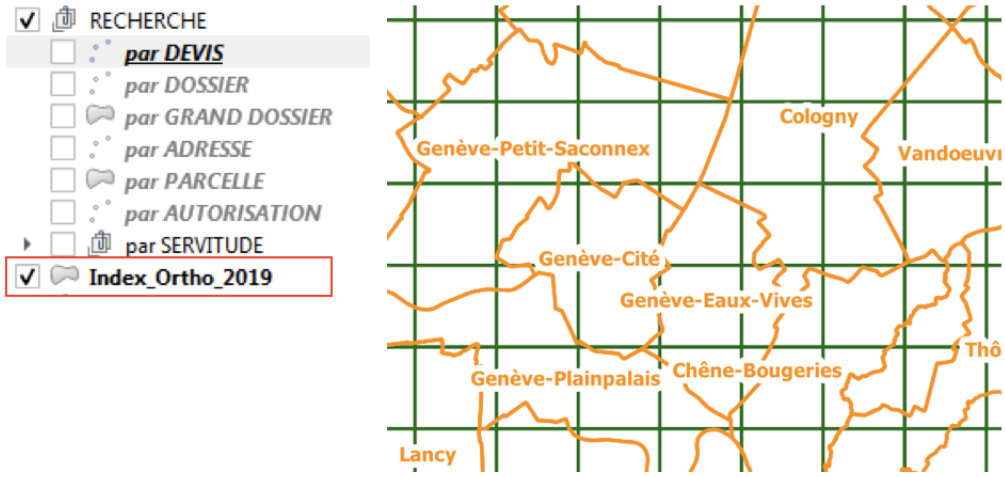


Figure 12 : Contour des tuiles orthophotos sous forme de grille shapefile, ayant la forme du canton de Genève.

Pour charger l'orthophoto, le principe est le même qu'auparavant :

-Soit **cocher et sélectionner** la couche « Index_Ortho2019 » puis cliquer sur le bouton d'action  et appuyer sur Orthophoto  . Ensuite il suffit de cliquer sur la carte à l'endroit souhaité, et l'orthophoto de cette zone va se charger.

-Soit **cocher** la couche « Index_Ortho2019 » et cliquer sur le bouton d'identification  , puis faire un clic droit à l'endroit souhaité sur la carte et choisir Orthophoto.

Il est évidemment possible de charger plusieurs orthophotos, en cliquant sur plusieurs endroits sur la carte (Figure 13).

Figure 13 : Carte du canton avec quelques orthophotos chargées.

Les orthophotos chargées vont se mettre automatiquement sous le groupe ORTHOPHOTOS (Figure 14).

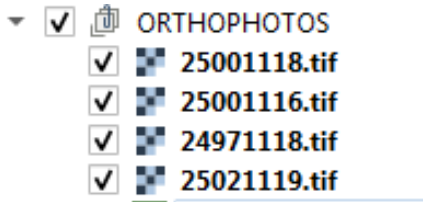


Figure 14 : Orthophotos chargées dans l'arborescence des couches sous le groupe ORTHOPHOTOS.

Il est préférable de les supprimer lorsque l'on en a plus l'utilité, car elles sont volumineuses. Il est possible de les supprimer soit :

- En activant le bouton d'action de la couche « Index_Ortho2019 »  et en cliquant sur l'orthophoto  , celle-ci disparaît comme elle apparaît.

- Manuellement en faisant un clic droit, supprimer.

- En fermant le projet sans l'enregistrer.

6c. Action pour charger la grille du modèle numérique de l'endroit cliqué

Cette action permet de charger une grille de points séparés de 2 mètres de distance, en affichant les altitudes des points, provenant du modèle numérique. Cette action a été implémentée sur les trois types de modèles numériques, *i.e.* le MNT, MNS et MNA.

Exemple pour charger la grille du MNT : C'est le même principe que pour charger l'orthophoto, sauf que cette fois, il faut sélectionner la couche « Index_MNT_2019 », qui se trouve tout en haut de l'arborescence. Puis cliquer sur  pour charger le shapefile contenant la grille de points. Le shapefile va se charger sous « MNT_grille_2m », qui se trouve dans le groupe « Modèles numériques ». Pour faire disparaître la grille, il suffit de sélectionner à nouveau la couche « Index_MNT_2019 » et de cliquer sur la grille que l'on souhaite enlever.

7. Google Street View

Pour ouvrir Google Street View, il suffit d'activer le bouton et de cliquer n'importe où sur la carte. Il faut ensuite cliquer sur l'icône encadrée en rouge dans la figure 15 pour obtenir la vue sur Google Maps.

Parfois, la page de Street View reste noire, dû au fait qu'il n'y a pas de Street View à cet endroit. Il est donc préférable d'utiliser l'action Google Maps implémentée dans QGis (point précédent).

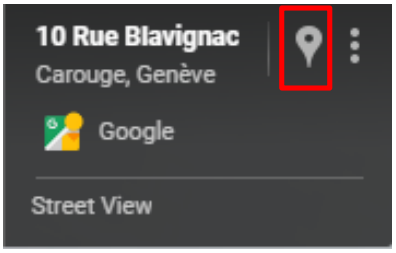


Figure 15 : Icône pour accéder à Google Maps.

Outils « Mise en page »

1. Mise en page prédéfinie

Pour ouvrir une mise en page type, il faut se rendre dans la barre d'outils personnalisée et cliquer sur le gestionnaire de mise en page  , puis double cliquer sur l'une des trois mises en pages type préenregistrées (Figure 16).

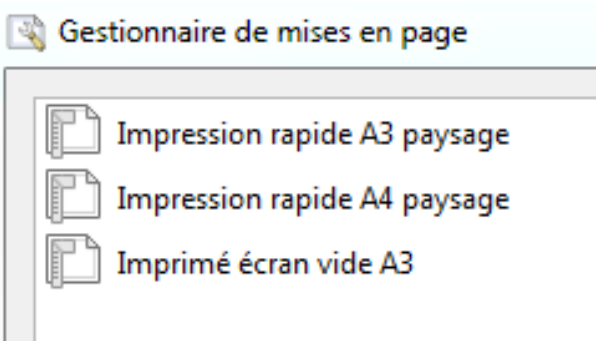


Figure 16 : Mises en page à choix dans le gestionnaire de mises en page.

Pour modifier la taille (A3/A4) et l'orientation (portrait/paysage) de la carte, il faut faire un clic droit sur la carte de la mise en page -> propriétés de la page (Figure 17).

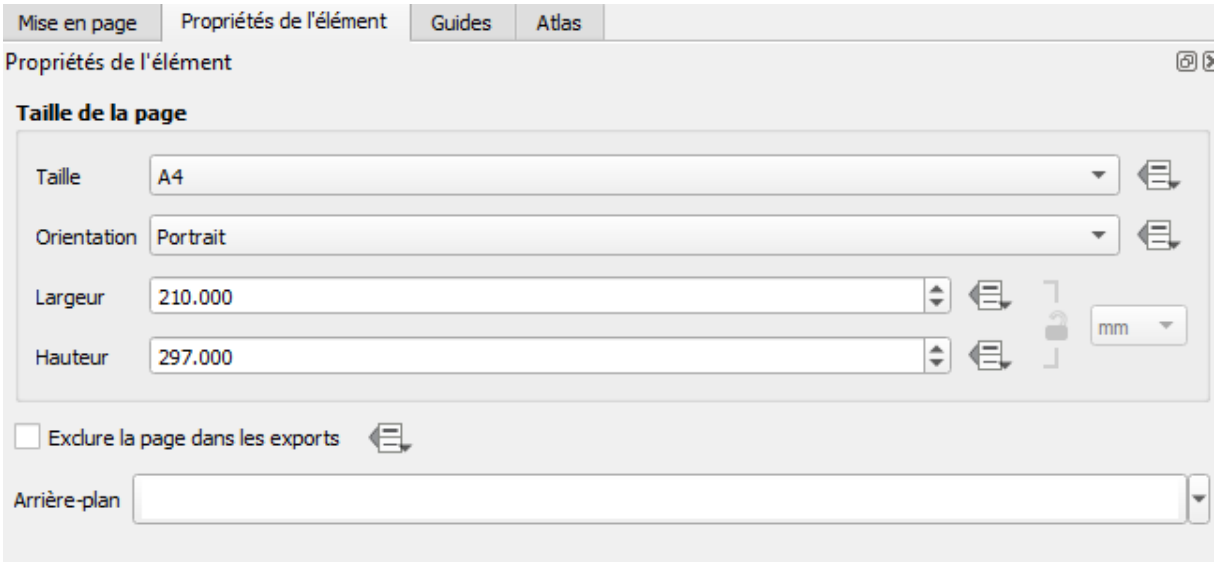


Figure 17 : Changement de taille et d'orientation dans les propriétés de la page.

1. Les mises en page « Impression rapide A3 paysage » et « Impression rapide A4 paysage » sont des mises en page contenant quelques informations utiles au quotidien. La mise en page apparaît sur la dernière vue de la carte (Figure 18) ; pour faire en sorte que la carte soit la même que celle dans le projet QGis, il faut sélectionner la carte dans la mise en page et aller dans « propriétés de l'élément » (Figure 19). Cliquer sur l'icône  « gérer l'emprise de la carte pour qu'elle corresponde à l'emprise du canevas principal ». Ce bouton va actualiser l'échelle, l'endroit sur la carte ainsi que l'affichage des couches du projet QGis dans la mise en page.

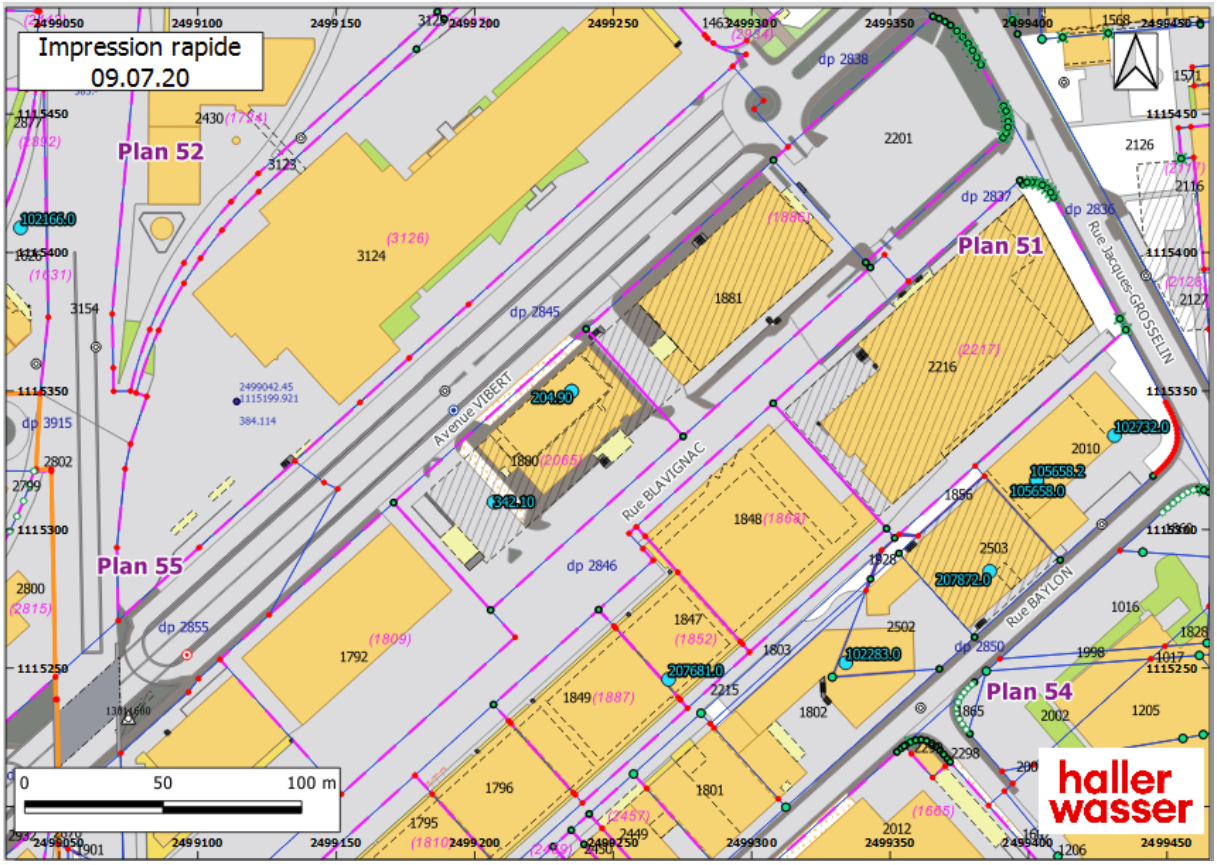


Figure 18 : Exemple de la mise en page « impression rapide A3 paysage ».

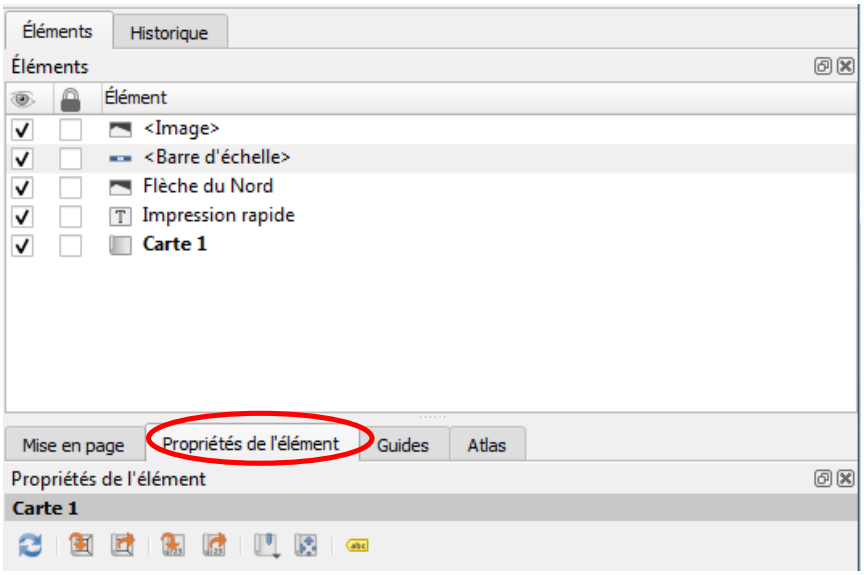


Figure 19 : Propriétés de la carte, sur la fenêtre de mise en page.

En revanche, si sur le projet QGis l'on sélectionne ou désélectionne des couches, il faudra alors cliquer sur le bouton de mise à jour de l'aperçu de la carte , qui se trouve lui aussi dans les propriétés de l'élément (Figure 19).

2. La mise en page « Imprimé écran vide A3 » ne contient que la carte, qu'il faudra rafraîchir avec le projet par le biais de la même icône qu'auparavant .

3. D'autres mises en page plus spécifiques ont été enregistrées en tant que modèles : « Type HW A3 paysage » et « Type HW A4 portrait », qu'il faudra charger d'une manière différente que précédemment. Il faut faire un Ctrl+P dans le projet QGis, afin de créer une mise en page vide (**NB** : il n'est pas nécessaire de lui attribuer un nom). Ensuite aller dans Mise en page -> Ajouter des éléments depuis un modèle (Figure 20). Le modèle se trouve dans le chemin suivant :

C:\Users\votre_nom_utilisateur\AppData\Roaming\QGIS\QGIS3\profiles\HW\composer_templates

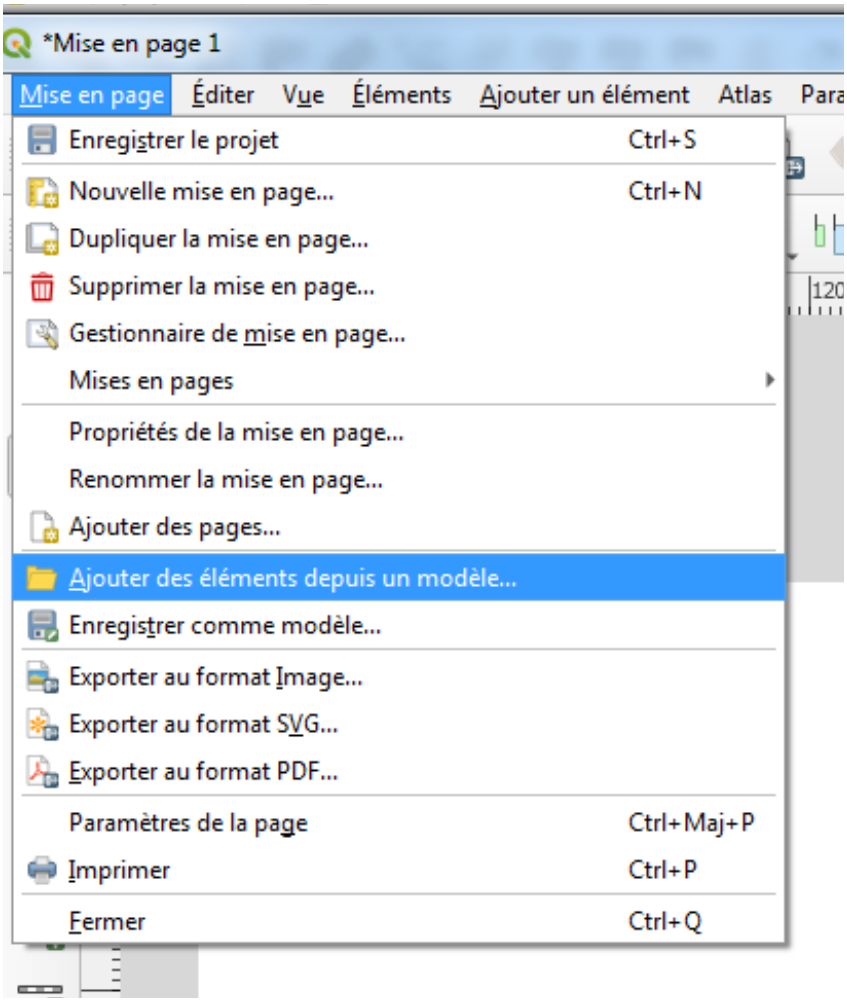


Figure 20 : Comment charger un modèle préenregistré à une nouvelle mise en page.

Ces mises en page contiennent plus de détails, qui sont utiles lors d'une impression de plan cadastral, esquisse, plan de situation etc. (Figures 21 et 22).

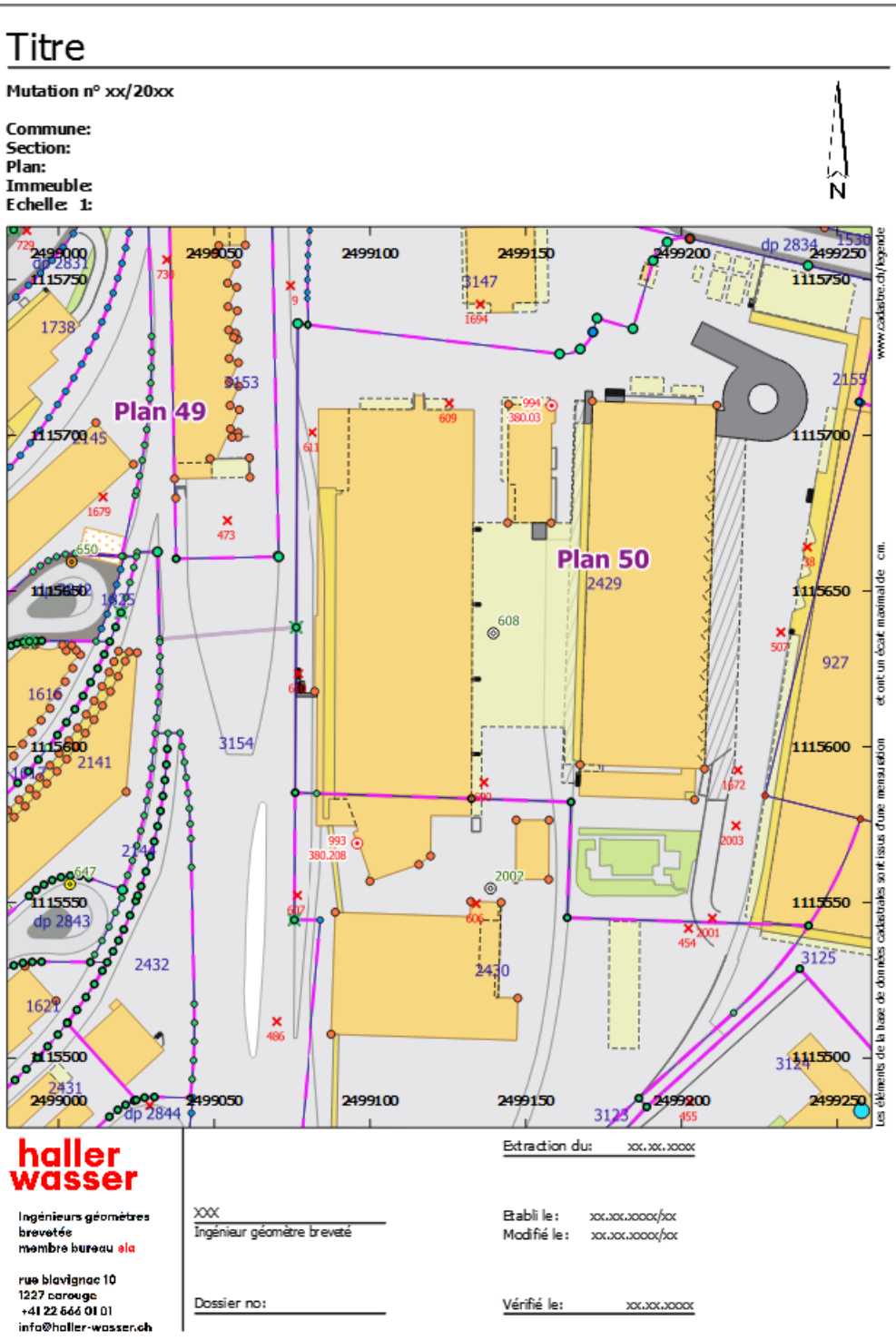


Figure 21 : Exemple d'impression de « Type HW A4 portrait ».

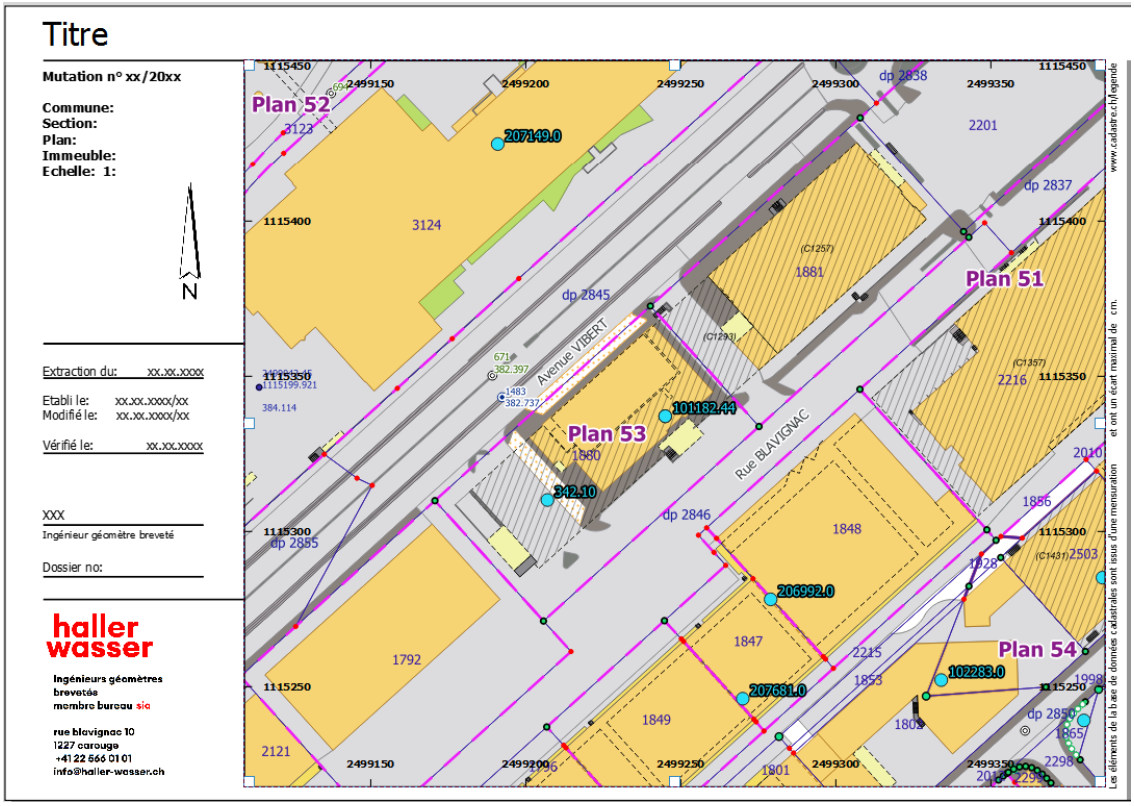


Figure 22 : Exemple d'impression de « Type HW A3 paysage ».

Il suffit de rajouter les informations propres au projet dans les zones de texte prédéfinies, qui se trouvent sous l'onglet « éléments », à droite de la mise en page (Figure 23). En cliquant sur l'une des zones de texte (celles en bas de liste, qui n'ont pas le cadenas coché), il sera possible de mettre le texte souhaité dans les « propriétés principales de l'élément » (Figure 24). **NB** : Il y a six zones de texte modifiables : Titre, Mutation, Extraction, XXX (nom de l'ingénieur géomètre breveté), Dossier no et type de mensuration + écart maximal (à droite de la carte).

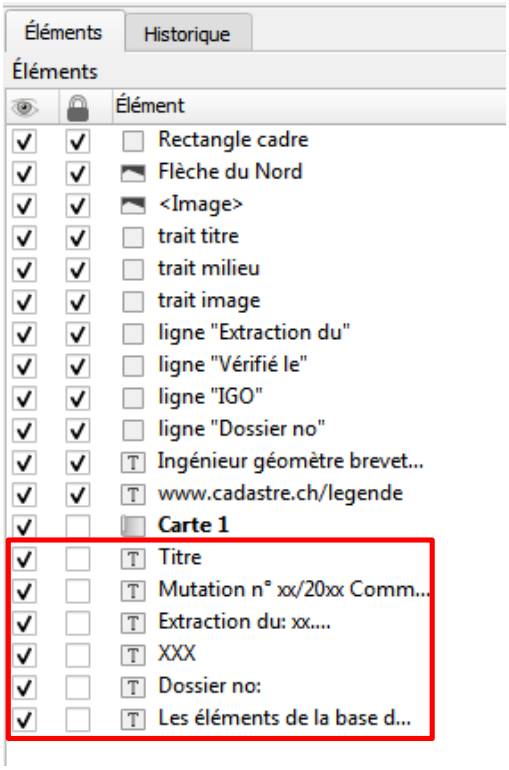
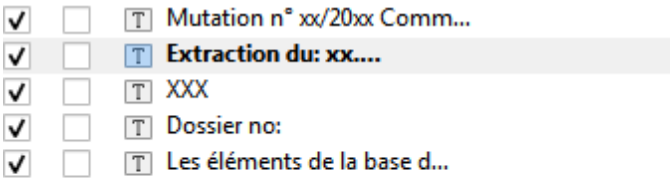


Figure 23 : Liste des différents objets de la mise en page, avec les zones de texte modifiables encadrées en rouge.



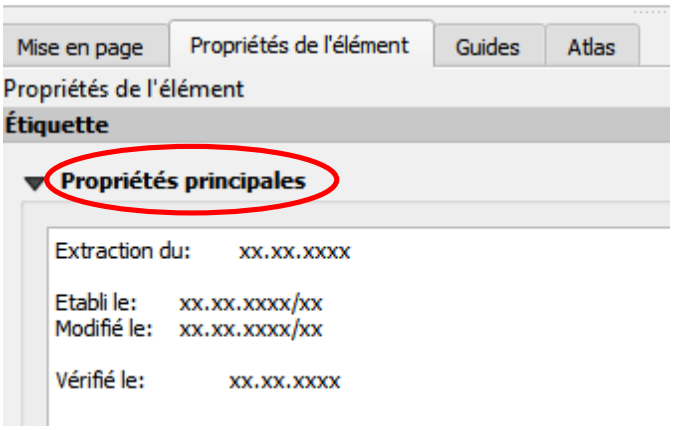


Figure 24 : Zone de texte modifiable.

Outils « Ajout de points »

1a. Introduire des altitudes et des coordonnées

Si l'on souhaite ajouter des points géo-référencés à un shapefile existant, il faut en tout premier lieu sélectionner la couche que l'on veut éditer dans l'arborescence des couches, puis cliquer sur le bouton qui permet de basculer en mode édition  . Ensuite, cliquer sur le bouton « Numerical Digitize »  . Ce bouton permet d'ajouter une nouvelle entité à la couche, en géo-référençant le nouveau point grâce à ses coordonnées GPS. Il suffit d'entrer les coordonnées **X** (2000000) et **Y** (1000000), en s'assurant que le « CRS of the Project » soit coché (Figure 25).

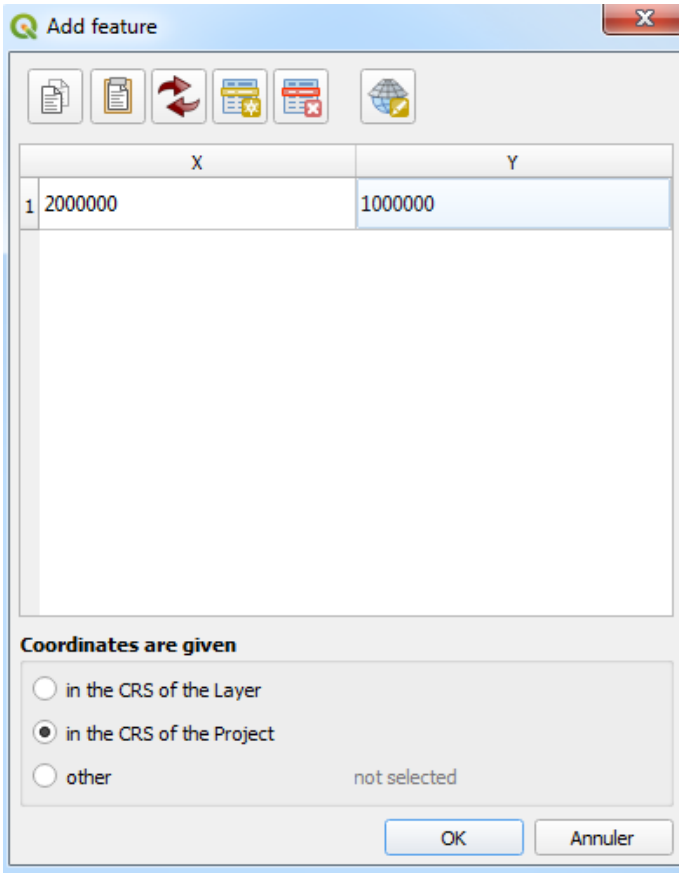


Figure 25 : Ajout d'entités à une couche, avec les coordonnées X et Y.

En appuyant sur Ok, il sera ensuite possible de remplir les champs de cette nouvelle entité, qui seront enregistrés dans la table attributaire de la couche (Figure 26). Il faut à nouveau remplir les champs X et Y, pour qu'ils apparaissent dans la table attributaire et dans l'étiquette.

NB : Le X et le Y sont inversés dans le logiciel de gestion du bureau (Spadice), il faut donc rentrer la valeur de 2000000 pour **Y** et 1000000 pour **X**.

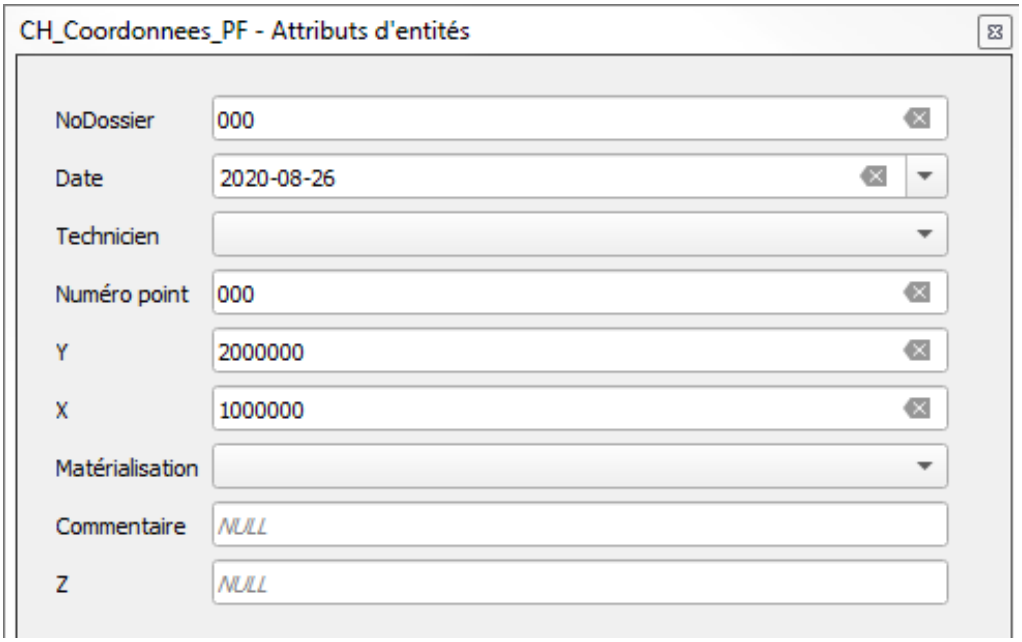


Figure 26 : Exemple de remplissage de champs pour un ajout d'entités à la couche.

Finalement, cliquer sur Ok, puis appuyer à nouveau sur le bouton d'édition  , et cliquer sur « enregistrer » pour que les nouvelles entités soient bien sauvées dans la couche.

Autre méthode

Le plugin « Lat Lon Digitize »  permet d'effectuer la même tâche, en choisissant la projection et l'ordre des coordonnées GPS (Figure 27). Le principe est le même que pour le plugin précédent ; il faut se mettre en mode édition, cliquer sur le plugin et ensuite remplir le formulaire avec les informations du nouveau point. Le problème ici, c'est que le point se génère deux fois. Une fois au bon endroit, et une fois placé au mauvais endroit. N'ayant pas réussi à résoudre ce mystère, nous avons préféré l'autre plugin pour notre utilisation.

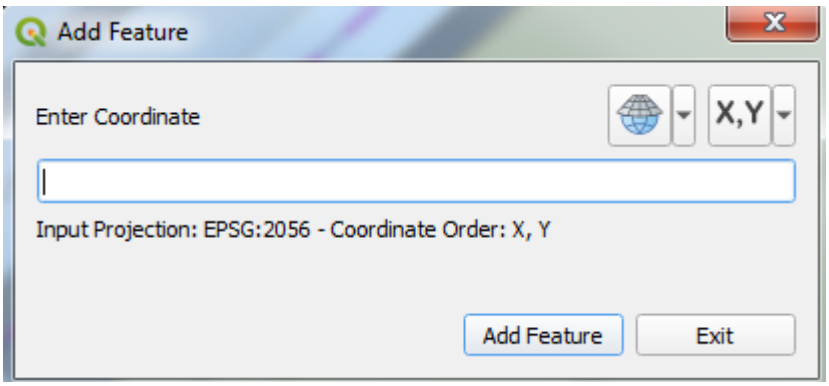


Figure 27 : Autre plugin d'ajout d'entités géo-référencées (Lat Lon Digitize).

1b. Renseigner des points fixes disparus (points du bureau)

Si l'on souhaite renseigner un PF (point fixe) disparu, il faut avant tout interroger le PF en question, par le biais du bouton d'identification  , afin de savoir quel est son numéro de dossier. Puis, dans

le but d'obtenir les coordonnées du point il faut faire une saisie de coordonnées  , et les copier. Puis, sélectionner la couche « CH_PF_Disparu » dans l'arborescence des couches, et faire les mêmes manœuvres que pour le point précédent, c'est-à-dire  +  , coller les coordonnées, rentrer les quelques informations demandées et appuyer à nouveau sur le  pour enregistrer les modifications.

Attention : cette couche sert à renseigner les PF disparus du bureau, et non ceux du cadastre. Pour que les points du cadastre soient mis à jour, il faut leur transmettre le point disparu.

1.5 Méthodes

1.5.1 Ajout de couches

Connexion au MapServer

La connexion se fait par l'ajout d'une couche ArcGIS MapServer. Il suffit d'entrer l'URL du MapServer et de cliquer sur « connexion » (Figure 28).

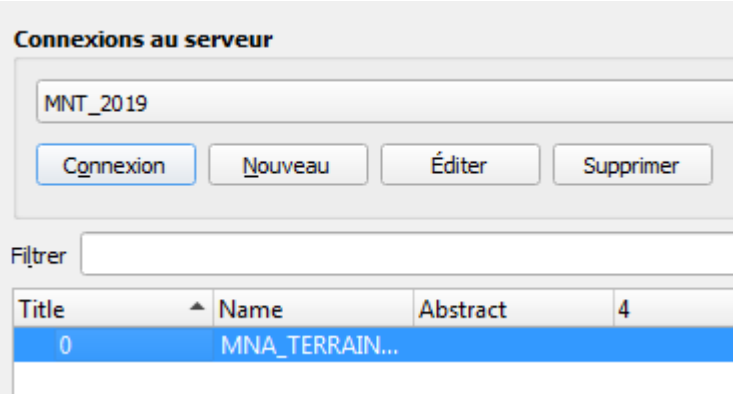


Figure 28 : Exemple de connexion à un MapServer, pour charger des modèles numériques dans le projet.

Cette manœuvre nous permet d'ajouter les modèles numériques (terrain, surface et hauteur) sous forme de raster, à notre projet. Ces couches apparaissent avec des teintes de gris, mais il est possible de leur attribuer une couleur (Figure 29b), afin de les différencier. Les rasters sont très peu personnalisables sur QGis, au contraire des vecteurs, dont la plupart des paramètres sont modifiables.

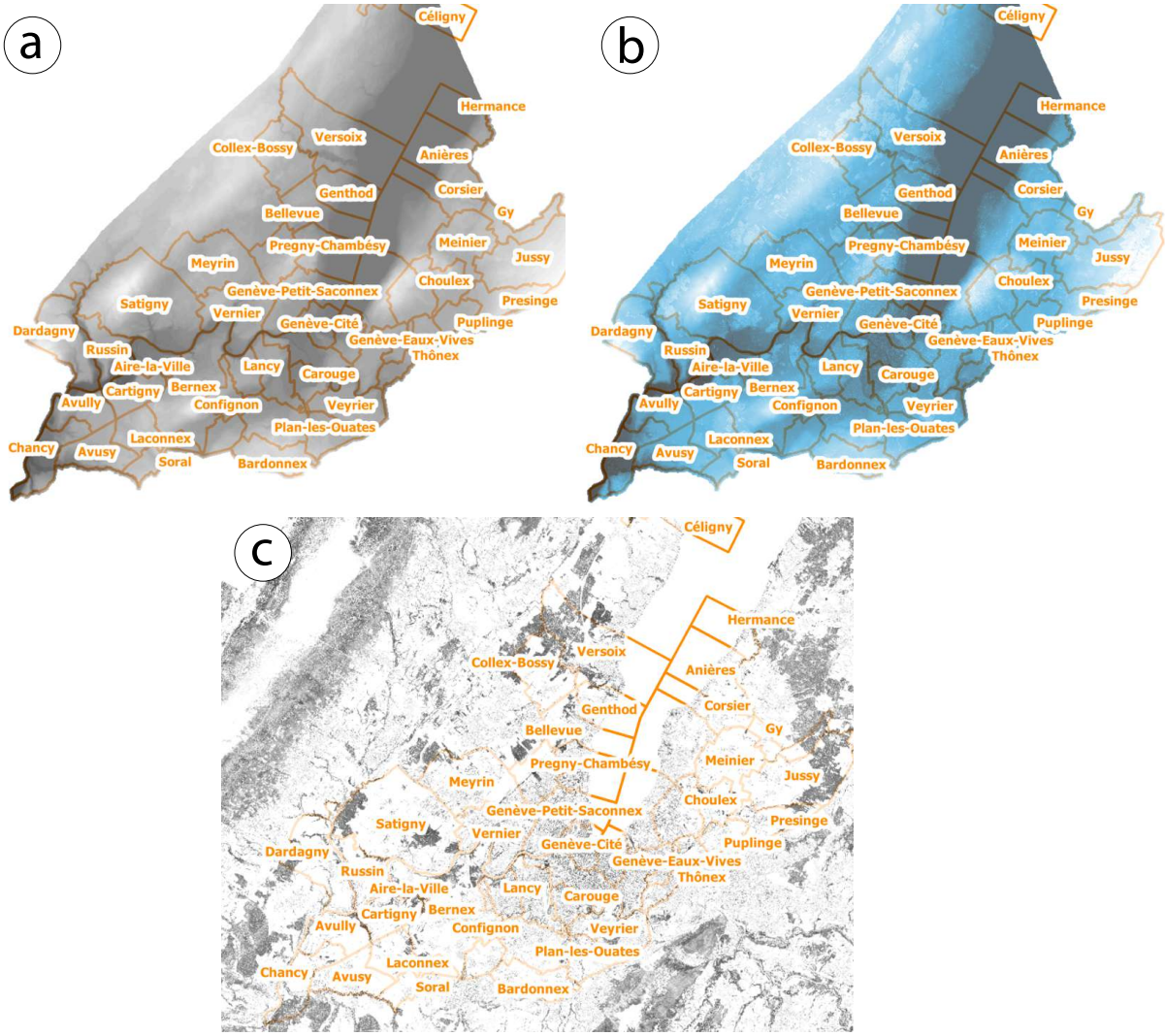


Figure 29 : Exemple de rasters pour **a)** le MNT 2019, **b)** le MNS 2019 et **c)** le MNA 2018.

URLs :

https://ge.ch/sitgags2/rest/services/RASTER/MNA_TERRAIN/MapServer

https://ge.ch/sitgags2/rest/services/RASTER/MNA_SURFACE/MapServer

https://ge.ch/sitgags2/rest/services/RASTER/MNA_HAUTEUR/MapServer

Ajouter une couche de texte délimité

Afin d'ajouter un fichier texte au projet, par exemple le fichier des devis ou des dossiers, il faut aller dans : Couche -> ajouter une couche -> ajouter une couche de texte délimité.

Les champs « X », « Y » et « Echantillon de données » se remplissent automatiquement lors du chargement du fichier texte. Le fichier texte en question utilise des tabulations comme délimiteur, c'est pourquoi nous avons coché « Tab » dans l'onglet « Format de fichier » (Figure 30).

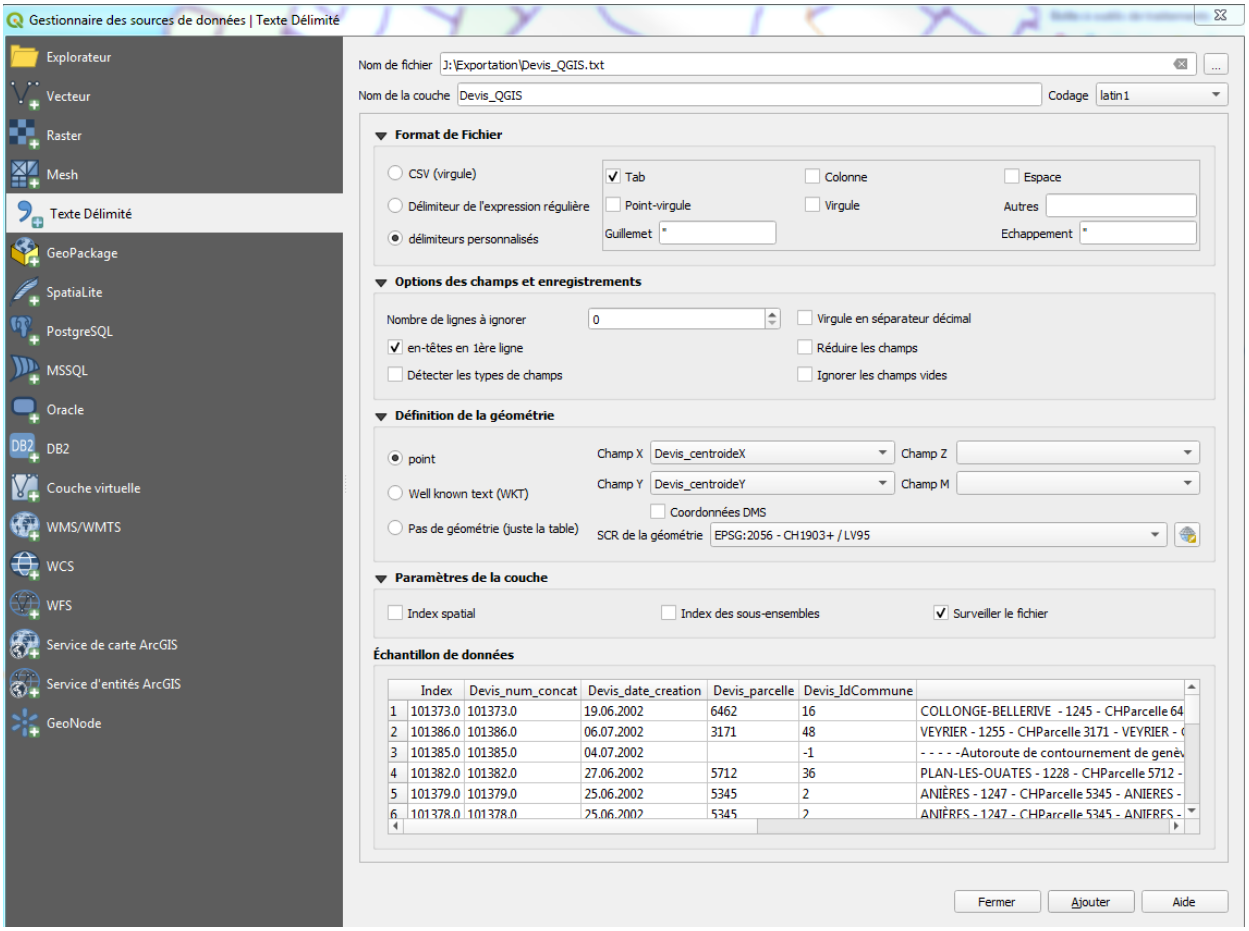


Figure 30: Exemple d'ajout de couche de texte délimité, avec les champs à remplir.

1.5.2 Charger des couches depuis un autre projet

Si l'on souhaite ajouter des couches qui ne se trouvent pas dans le projet actuel, notamment parce qu'elles sont trop volumineuses, il suffit de sélectionner le groupe dans l'arborescence sous lequel on souhaite charger les couches, puis d'aller dans Couche -> intégrer des couches et des groupes (Figure 31). Dans notre cas de figure, nous souhaitons charger des RDPPF, qui ont été intégrés à un projet annexe par le biais d'une connexion au MapServer du SITG.

Le cadastre des restrictions de droit public à la propriété foncière (cadastre RDPPF) est un système d'information officiel qui récapitule les principales restrictions de droit public à la propriété foncière (www.ge.ch). Le cadastre RDPPF comporte de nombreuses couches, volumineuses, qui ralentiraient passablement le projet de base si on les chargeait par défaut.

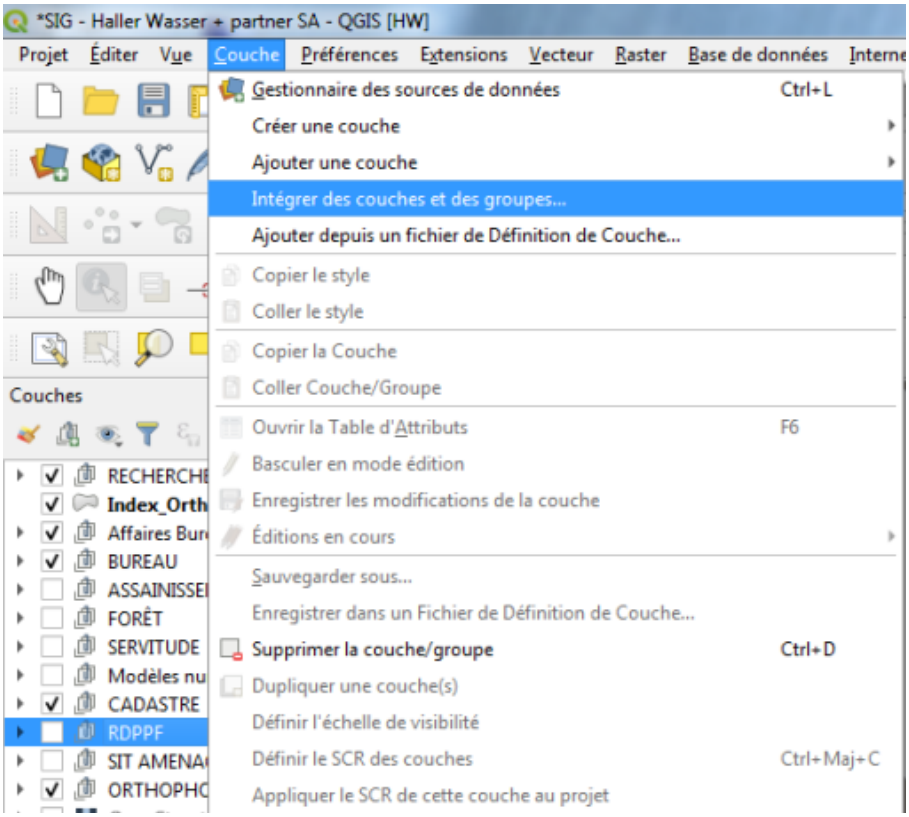


Figure 31 : Ajout de couches provenant d'un autre projet, spécifique aux RDPPF.

Ensuite, il faudra charger le projet « projet_RDPPF », où il sera possible d'ajouter les couches souhaitées dans le projet (Figure 32), et ces couches apparaîtront directement dans le groupe RDPPF. En fermant le projet sans l'enregistrer, ces couches disparaîtront du projet.

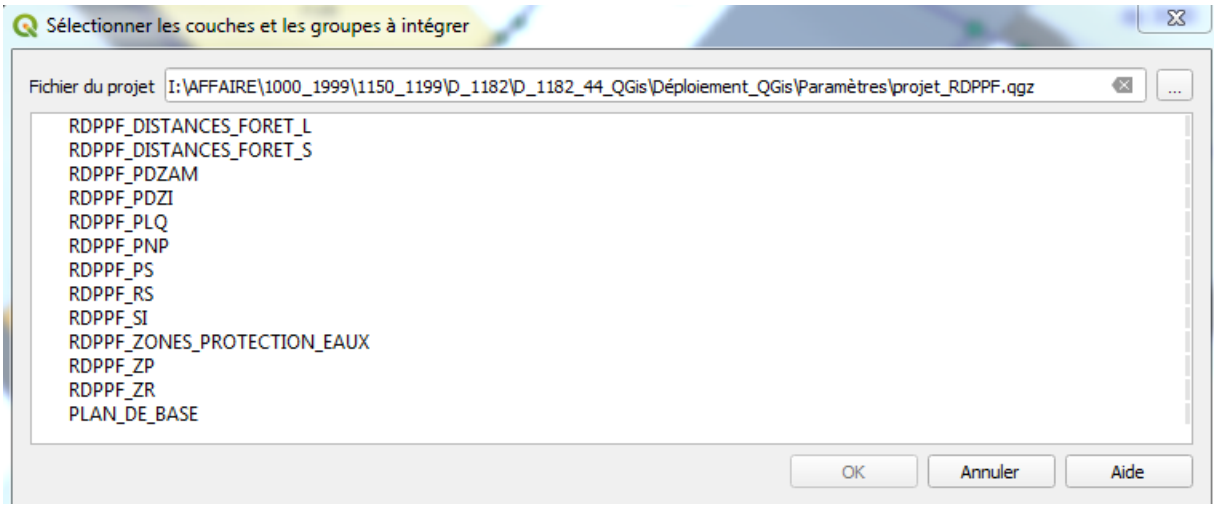


Figure 32 : Liste des couches RDPPF disponibles dans le projet annexe.

1.5.3 Création de formulaires

La création de formulaires permet de mieux classer les différents attributs d'une couche. Pour ce faire, il faut se rendre dans les propriétés de la couche -> formulaires d'attributs, et sélectionner « conception par glisser/déposer ». Il est possible d'enlever certains champs (bouton), qui n'ont

pas d'intérêt particulier pour le formulaire souhaité. Dans la figure ci-dessous, nous avons décidé de ne garder que les champs « COMMUNE » et « NO_PARCELL » (Figure 33).

Le bouton  permet de créer :

1. **Un onglet**, comme par exemple « Recherche par parcelle », qui sera le conteneur principal où seront stockés les groupes créés successivement.
2. **Un groupe dans un conteneur**, comme par exemple « Commune » et « Parcelle », où seront stockés les différents champs de la table attributaire (Fields).

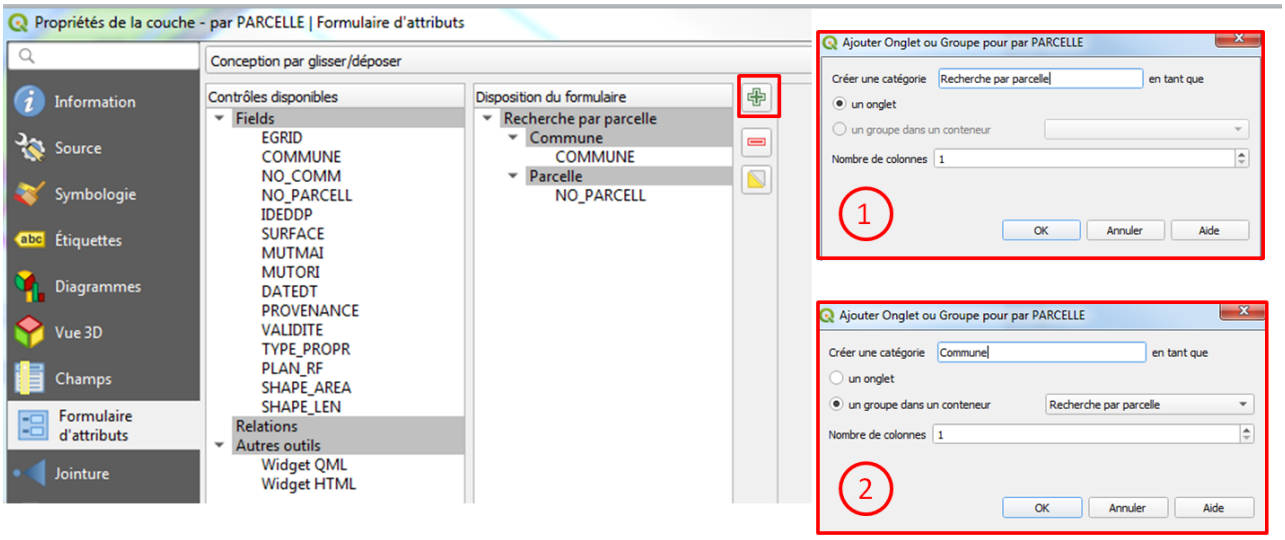


Figure 33: Exemple de création de formulaire, dans les propriétés de la couche.

Les formulaires apparaissent notamment lorsque l'on fait une recherche d'entité par valeur, ou quand l'on rajoute une nouvelle entité (géo-référencée ou non) à un shapefile existant. Cela permet une visibilité plus nette et une catégorisation des informations pertinentes.

1.5.4 Enregistrer un style au format QML

Lorsque l'on définit un style particulier à une couche, ce style peut être enregistré et appliqué à d'autres couches, sans devoir refaire les mêmes manipulations. Dans un fichier style, la plupart des propriétés de couches sont modifiables, et il est possible de choisir quelles catégories de style nous allons enregistrer, en les cochant/décochant (Figure 34).

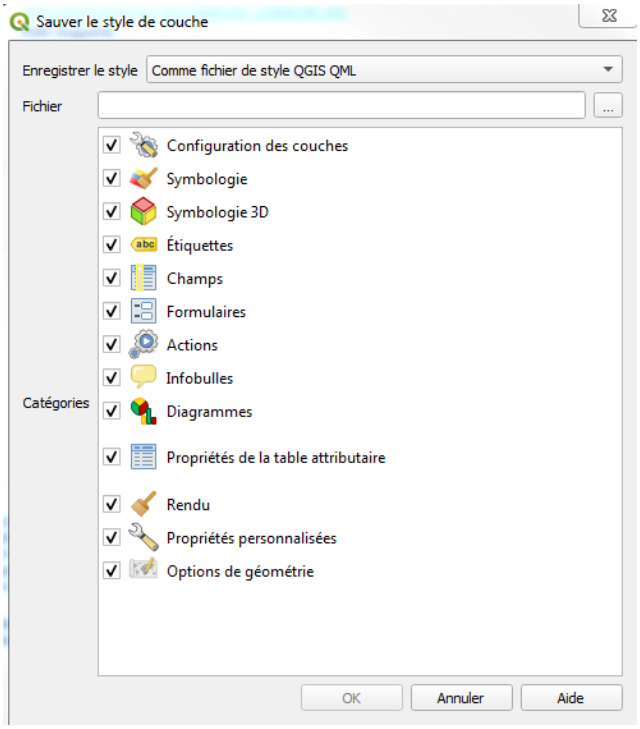


Figure 34: Liste des propriétés modifiables d'une couche.

Exemple : nous avons changé la symbologie, le type d'étiquettes et l'échelle d'affichage (dans onglet rendu) de notre couche, et nous aimerions appliquer les mêmes paramètres à une autre couche. Il suffit de cliquer sur « style » (en bas de fenêtre), enregistrer le style, et de l'enregistrer au format .qml (Figure 35).

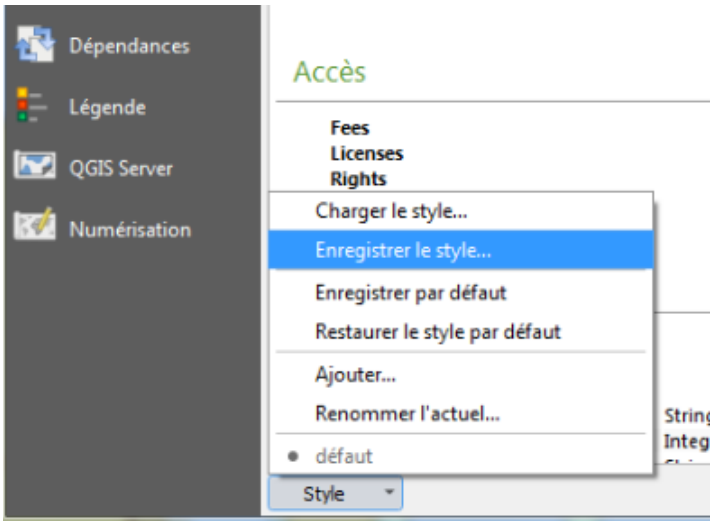


Figure 35: Enregistrement du style défini pour une couche.

1.5.5 Gestion de l'affichage lors de l'utilisation de l'outil « identifier l'entité »

Il est possible de déterminer quel attribut de la couche va apparaître en premier lors d'une identification d'entité. Pour ce faire, il faut se rendre dans la table attributaire de la couche, changer l'affichage en bas à droite pour se mettre en vue formulaire (et non en vue tabulaire qui est la vue

par défaut), puis sélectionner expression -> prévisualisation de colonne, et choisir l'attribut désiré (Figure 36).

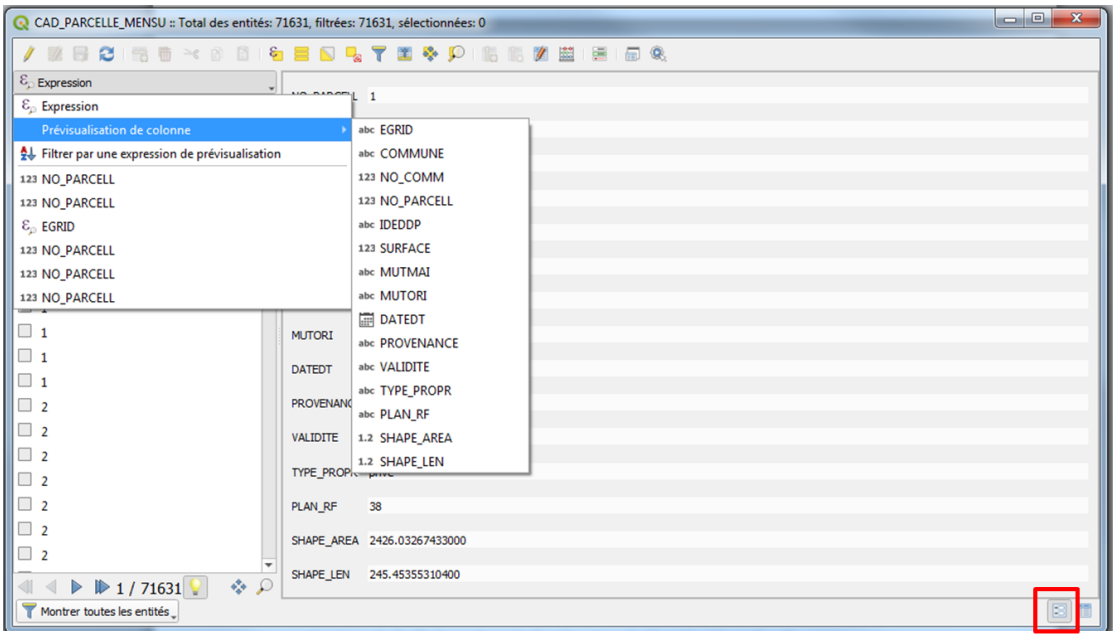


Figure 36: Définition de l'attribut à afficher via la table attributaire de la couche.

1.5.6 Mise en page

Créer soi-même sa mise en page

Pour créer une mise en page personnalisée, il faut aller dans Projet -> Nouvelle mise en page, ou bien faire le raccourci Ctrl+P.

Sur la page de la mise en page, cliquer sur :

- Ajouter une nouvelle carte à la mise en page  : pour faire apparaître la carte qui se trouve dans le projet QGis. **NB :** l'échelle n'est pas exactement la même que sur le projet QGis. Si l'on souhaite avoir une échelle bien spécifique, il faudra la modifier dans la mise en page directement.

- Sélectionner/déplacer un élément  : pour déplacer la carte ou d'autres éléments (légende, barre d'échelle, etc.).

- Déplacer le contenu de l'élément  : pour ajuster la mise en place de la carte (utile pour des petits déplacements).

- Ajouter une légende à la carte 

- Ajouter une barre d'échelle à la carte 

- Ajouter une flèche du nord 

- Ajouter une zone de texte (pour écrire un titre par exemple) 

Personnalisation de la légende

Il est possible de personnaliser la légende qui va apparaître dans la mise en page, en choisissant quelles couches faire apparaître. Pour ce faire, il faut sélectionner « Légende » dans l'onglet « Eléments » (Figure 37a). Puis, dans les « Eléments de légende » qui se trouvent dans l'onglet « Propriétés de l'élément » (Figure 37b), il faut décocher « mise à jour auto » (Figure 37c), pour pouvoir ajouter/supprimer des couches  . Il est également possible de renommer les couches de la légende, en double cliquant sur la couche dans l'onglet « Eléments de légende » (Figure 37c). Il sera ainsi possible de donner des noms plus spécifiques aux couches.

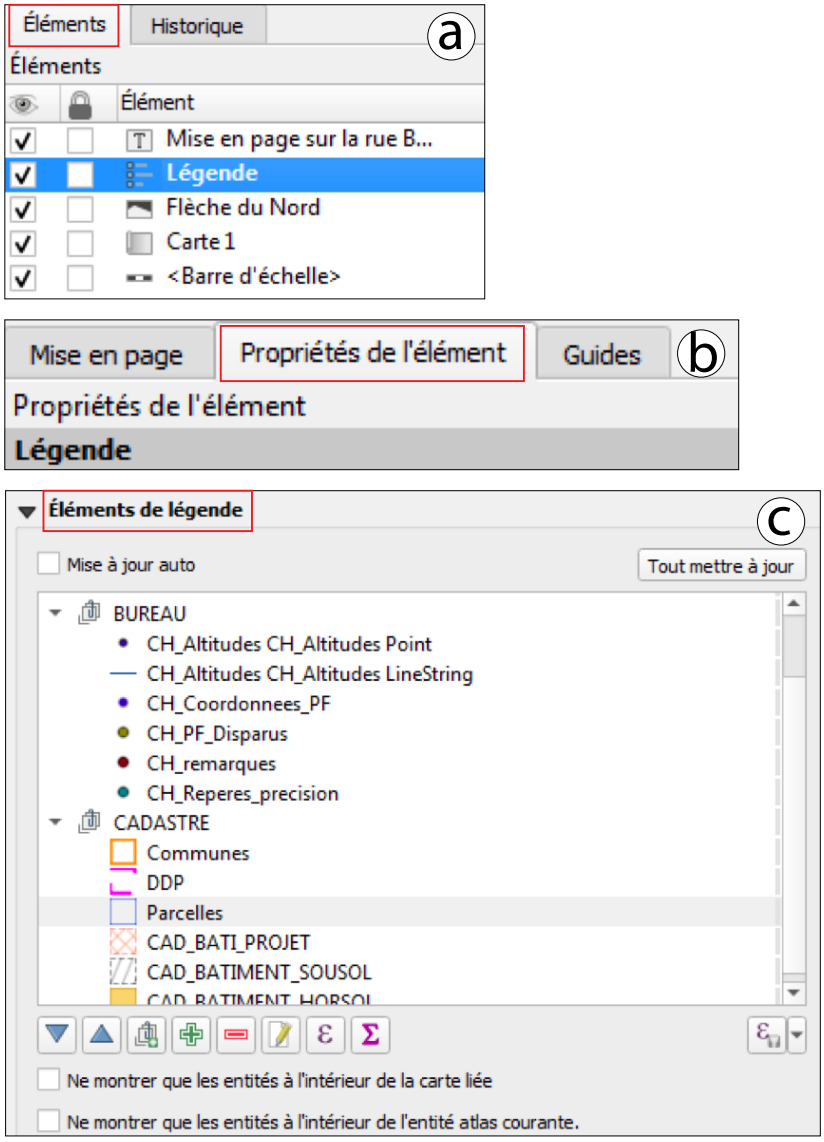


Figure 37 : Personnalisation de la légende : **a)** Onglet « Eléments », **b)** Onglet « Propriétés de l'élément » et **c)** Onglet « Eléments de légende ».

1.5.7 Symbologie avec règles

Pour que les entités sélectionnées dans le projet QGis (*par exemple* : une ou plusieurs parcelles) apparaissent également sélectionnées dans la mise en page et donc à l'impression, il faut aller dans les propriétés de la couche en question, et créer une symbologie basée sur un ensemble de règles. Il faut créer deux règles ; une règle « ELSE », qui sera la symbologie des parcelles non sélectionnées, et une règle contenant un filtre. Pour cette dernière, il faudra rentrer l'expression suivante : *is_selected($currentfeature)* dans le constructeur de chaîne d'expressions  .

Cette règle s'applique lorsque l'on sélectionne une parcelle, et la règle précédente s'applique aux parcelles non sélectionnées, *i.e.* le reste. Finalement, il faut déterminer une symbologie pour les entités sélectionnées et une pour les non-sélectionnées.

Pour sélectionner des entités, il faut avant toute chose sélectionner dans l'arborescence la couche « CAD_PARCELLE_MENSU », puis cliquer sur le bouton de sélection d'entités  . Il suffit ensuite de cliquer sur la parcelle désirée, pour que l'entité apparaisse sélectionnée (Figure 38a). Pour sélectionner plusieurs parcelles, garder la touche « shift » appuyée lors de la sélection. Pour désélectionner la ou les entités il faut cliquer sur « Désélectionner toutes les entités »  .

Dans le projet QGis, la couleur de sélection (rectangle rempli en cyan) (Figure 38a) a été déterminée dans les préférences du logiciel. La symbologie créée pour les parcelles sélectionnées n'apparaît pas, car elle est cachée par la sélection. En revanche, dans la mise en page, c'est la symbologie définie qui apparaît (rectangle avec contour rouge) (Figure 38b).

De plus, dans la mise en page, si l'on souhaite afficher la date du jour dans le titre, il faut noter : [% format_date(now(), 'dd.MM.yy')%].

Il est aussi possible de rajouter une table d'attributs à la mise en page  , et de définir les attributs que l'on souhaite afficher et dans quel ordre. Ainsi, pour les parcelles sélectionnées, les attributs choisis apparaitront à l'impression (Figure 38b).

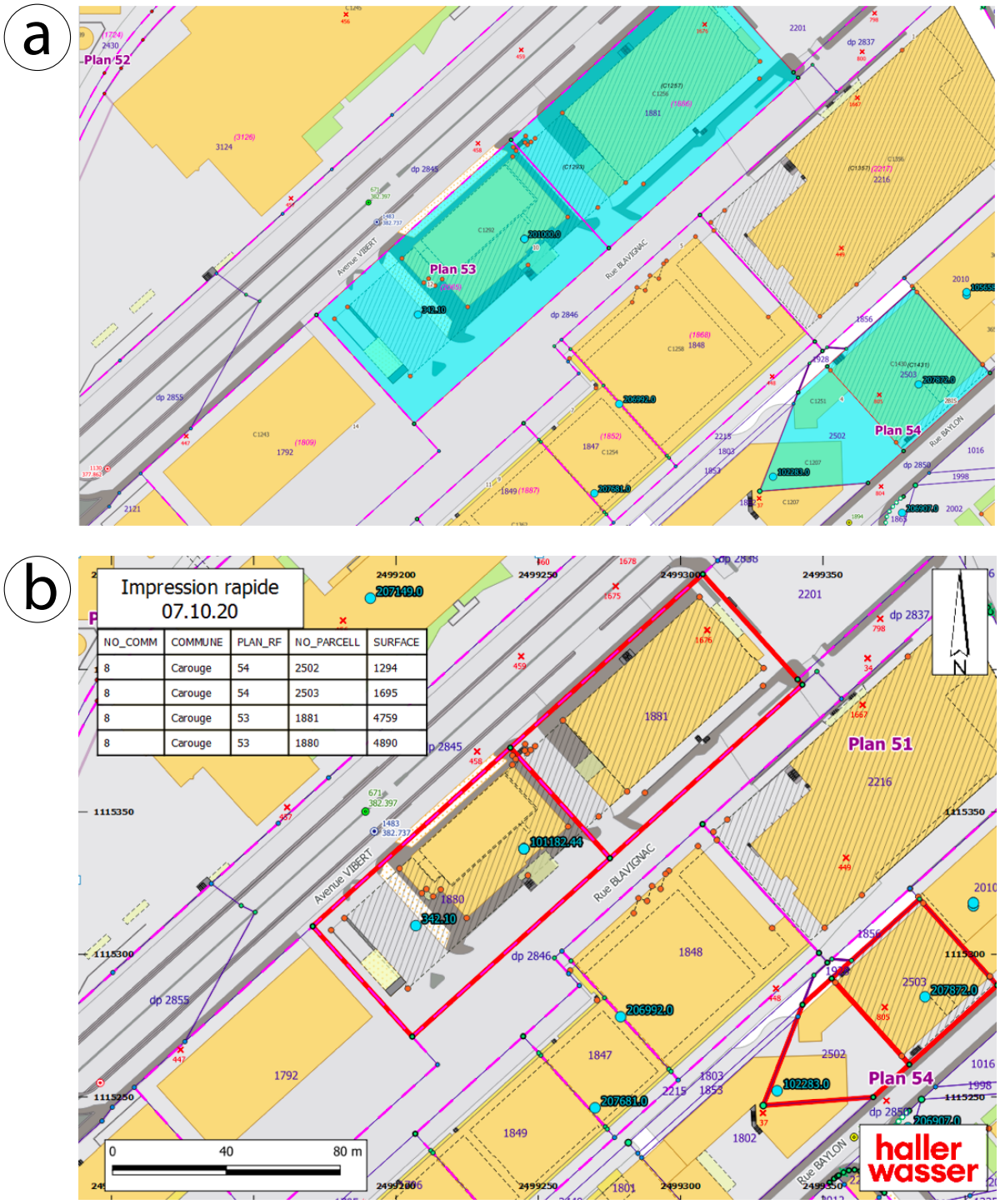


Figure 38 : **a)** Exemple de sélection d'entités dans la couche des parcelles et **b)** affichage de ces entités dans la mise en page, avec les attributs propres aux entités sélectionnées.

1.5.8 Etiquettes avec règles

Des étiquettes basées sur des règles ont été implémentées sur la couche des points limites, pour afficher le numéro du point, le numéro de mutation (MUTNUM) et le numéro de la mutation d'origine (MUTORI), ces derniers devant apparaître à la ligne et séparés par une barre oblique, comme ci-dessous (Figure 39).

Figure 39 : Etiquette des points limites.

Deux catégories d'étiquettes ont été créées : « MUTNUM pas bon » et « MUTNUM bon », qui contiennent des sous-étiquettes, dans le cas où le MUTORI serait pas bon, et dans tous les autres cas. Le numéro de point (NO_POINT) apparaît systématiquement, en revanche, les MUTNUM et MUTORI sont parfois inexistants ou égaux à 0. C'est pourquoi nous souhaitons afficher un tiret lorsque le MUTNUM ou le MUTORI sont soit : nuls, égaux à 0, 000000, 0000000 ou 00000000 (Figure 40).

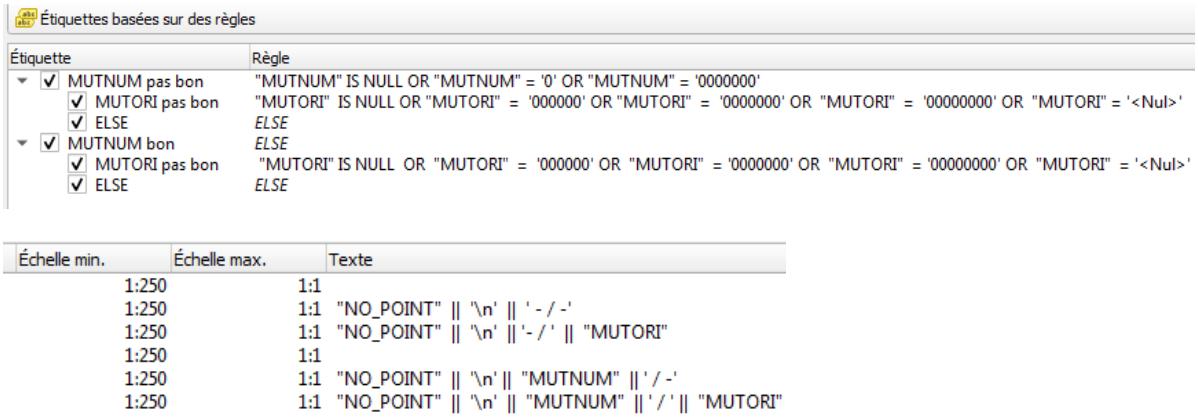


Figure 40 : Ensemble de règles imposées à l'étiquette des points limites.

D'autres étiquettes basées sur des règles ont été implémentées sur les servitudes, dans le but d'afficher le numéro de servitude avec le préfixe « RS », ainsi que la classe de servitude. Dans l'expression de l'étiquette, on rentre le code suivant :

concat('RS', ' ', "N0_REGISTE" , '\n' , replace (lower("CLASSE_SER"), '. Contenu : voir registre foncier', ' '))

La fonction « concat » permet de créer une chaine de caractères, qui peut contenir d'autres fonctions. Pour effectuer un retour à la ligne, il suffit d'intégrer le retour chariot '\n'. Le champ « CLASSE_SER » est en majuscules, c'est pourquoi on emploie la fonction « lower », pour tout mettre en minuscules. De plus, dans ce champ, après chaque classe apparaît le texte « Contenu : voir registre foncier ». En utilisant la fonction « replace », on peut changer ce texte par un espace (' '), pour que l'étiquette ne contienne que les informations pertinentes.

Il est aussi possible d'utiliser le « double pipe » (||) pour concaténer une chaîne dans QGis :

'RS' || ' ' || "N0_REGISTE" || ' ' || '\n'|| replace (lower("CLASSE_SER"), '. Contenu : voir registre foncier', ' '

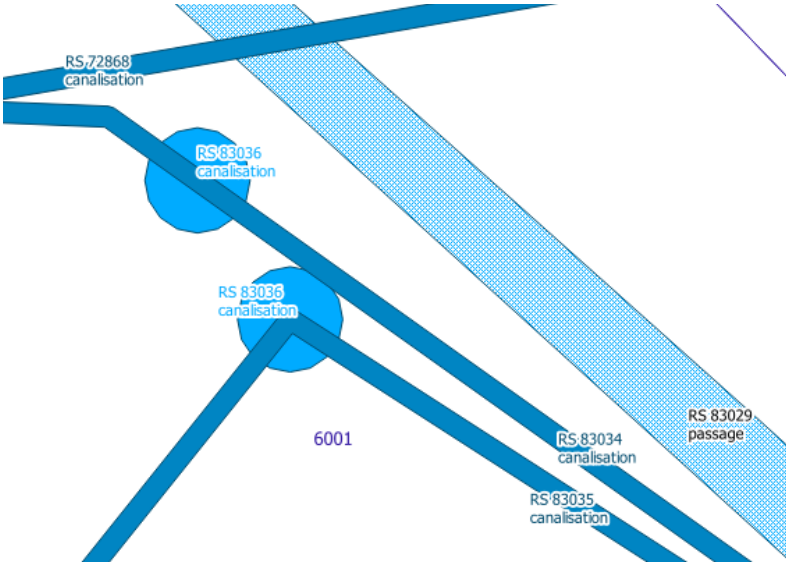


Figure 41 : Etiquettes des trois types de servitudes (point, ligne et surface).

1.5.9 Création d'infobulles

Les infobulles permettent d'afficher les informations souhaitées sur une entité, relatives à une couche. Le code s'écrit en langage HTML, ce qui permet de paramétrer divers éléments, tels que la taille et la couleur du cadre, le type de police (gras, italique etc.) et les champs à afficher. Il est également possible d'imposer des règles à des champs, notamment par le biais de « if », qui permet d'afficher tel ou tel valeur en fonction de l'attribut.

Exemple de code HTML pour les points limite :

```
<!DOCTYPE html>
<html>
<div class="section">
<b> <U>POINT LIMITE</b></U>
<ul>
<li><b> N°: </b>[%  concat(  "NO_COMM" ,' ', "NO_PLAN" , ' ', "NO_POINT" ) %]</li>
<li><b> Nature:</b> [%"NATURE"%] </li>
<li><b> Valeur:</b> [% "VALEUR" %] </li>
<li><b> Mutnum:</b> [% "MUTNUM" %]</li>
<li><b> Mutori:</b> [% "MUTORI" %] </li>
<li><b> Precplan: </b> [% "PRECPLAN" %] ([%  if(  "FIABPLAN"= 'oui' , 'Fiable', 'Pas fiable' )  %])  </li>
<li><b> X:</b> [%"X"%] </li>
<li><b> Y:</b> [%"Y"%] </li>
</ul>
</div>
<style>
      .section {
                        position: relative;
                       width: 230px;
 padding: 15px;
 box-sizing: border-box
                        text-align: justify;
       color: black;
                        background-color:#F3F3F2;
                        }
```

```
                    *{
  font-size: 12px;
  font-family: Arial;
  }
</style>
</html>
```

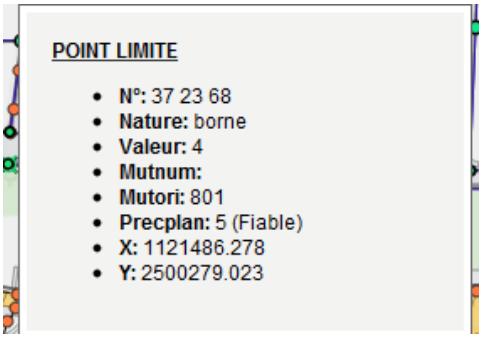


Figure 42 : Exemple d'infobulle pour les points limite.

Eléments du code

** ** : la balise définit une division ou une section dans un document HTML.
<b> : permet de mettre le texte en gras.
<U> : permet de souligner le texte.
<ul> : représente une liste d'éléments sans ordre particulier.
<li> : représente un élément dans une liste.
If : expression qui indique le bloc de code à exécuter si une condition spécifiée est vraie.
([% if("FIABPLAN"= 'oui' , 'Fiable', 'Pas fiable') %]) : Si l'entité dans le champ FIABPLAN est oui, afficher Fiable, sinon afficher Pas fiable.
<style> : permet de définir le style des éléments du code. C'est dans cette partie que l'on définit la taille du cadre (*width*), sa couleur, l'alignement du texte, la taille et la police.

1.5.10 Création d'actions

Ouvrir une page web

Il est possible de créer une action qui va ouvrir une page web en cliquant quelque part sur la carte. Il s'agit d'une action de type « ouvrir », dans laquelle il va falloir rentrer l'URL de la page web ainsi que les champs qui permettront de cibler l'ouverture de la page sur un endroit bien précis.

Par exemple, pour ouvrir le site du SITG à l'endroit cliqué sur la carte, il faut implémenter une action sur la couche des parcelles et noter l'URL suivant :

https://map.sitg.ch/?locparcelle=[%IDEDDP%]&mapresources=CADASTRE

Ici, l'IDEDDP est une concaténation du numéro de la commune et du numéro de la parcelle, un attribut déjà existant dans la couche des parcelles, pour que le SITG s'ouvre au même endroit que le clic.

Dans le même principe, l'action permettant d'ouvrir l'extrait foncier d'une parcelle ou d'un DDP devra contenir dans son URL les attributs du numéro de commune et numéro de parcelle/DDP pour afficher les informations relatives à la parcelle ou au DDP sélectionné.

https://ge.ch/terextraitfoncier/rapport.aspx?commune=[%NO_COMM%]&parcelle=[%NO_PARCELL%]

Nous avons également défini une action pour ouvrir la page web de SwissTopo (www.map.geo.admin.ch) sur l'orthophoto 2020 à 10 cm, qui sont consultables pour la Suisse entière. L'action qui nous permet d'y accéder a été implémentée sur la couche des parcelles. Dans l'URL, l'orthophoto est chargée en fonction des coordonnées GPS (E et N) en MN95. Comme la couche des parcelles ne contient pas de champ spécifique pour les coordonnées X ou Y, nous avons utilisé le calculateur d'expression, afin d'insérer une expression qui calcule le X et le Y au centre du polygone (de la parcelle) dans l'URL.

E= [% x (centroid($geometry))%]

N= [% y (centroid($geometry))%]

https://map.geo.admin.ch/?bgLayer=ch.swisstopo.swissimage&lang=fr&topic=ech&E=[% x(centroid($geometry))%]&N=[% y(centroid($geometry))%]&zoom=13

Ouvrir un fichier

Nous avons décidé de créer une action pour ouvrir la documentation pdf sur le projet QGis, pour que les utilisateurs puissent la consulter facilement, à tout moment. Afin d'ouvrir un fichier lorsque l'on clique sur une couche, il faut implémenter une action de type Python, et écrire le code suivant :

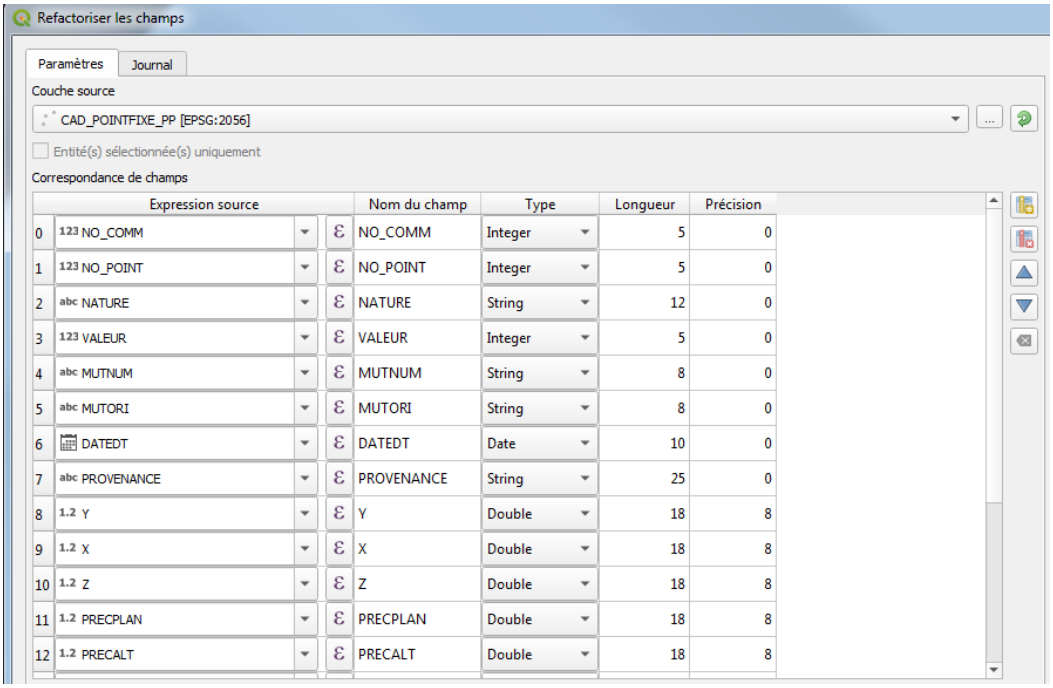
```
from os import startfile, path # On charge le paquet os qui va nous proposer la fonction startfile(), et path pour tester si un fichier existe.
from qgis.gui import QgsMessageBar
from qgis.utils import iface # On charge QgsMessageBar et iface pour afficher un message d'erreur.

proj = QgsProject.instance() # On stocke dans la variable proj l'instance du projet qui est ouvert.
path = "I:/Remplacement_MapInfo_QGIS.pdf" # Dans la variable path, on complète ce chemin avec le nom du répertoire qui contient le fichier pdf.
if os.path.isfile(path): # On teste l'existence du fichier.
      startfile(path) # On ouvre ce fichier grâce à la commande startfile.
else :
      iface.messageBar().pushMessage("Attention le fichier "\"+path+"\" n'existe pas.\nL'ouverture est impossible.", level=QgsMessageBar.CRITICAL, duration=0) # On affiche un message d'erreur.
```

1.5.11 Outils de traitement

L'outil de traitement « Refactoriser les champs » est un algorithme qui permet de créer une nouvelle couche, déduite d'une couche existante, proposant une modification des types de champs (*integer*, *real*, *double*, *string*, *date*, etc.) (Figure 43). Il est possible de modifier le type de tous les champs présents dans une couche, et cela peut s'avérer utile lorsque l'on a par exemple un type *double*, qui va donner plusieurs chiffres après la virgule, alors qu'on souhaite avoir un nombre entier. En cliquant sur exécuter, un couche temporaire refactorisée va se créer, qu'il faudra ensuite exporter en shapefile, pour pouvoir l'enregistrer définitivement dans le projet.

Nous avons refactorisé les couches propres au bureau, qui provenaient de MapInfo. D'une part, ces couches étaient au format .tab, qui n'était pas en adéquation avec le reste des couches du projet, et d'autre part, certains champs avaient un type qui ne convenait pas, donc nous l'avons modifié. En revanche, il est inutile de modifier les couches provenant du cadastre, car la mise à jour écrasera les modifications.



	Expression source	Nom du champ	Type	Longueur	Précision
0	123 NO_COMM	NO_COMM	Integer	5	0
1	123 NO_POINT	NO_POINT	Integer	5	0
2	abc NATURE	NATURE	String	12	0
3	123 VALEUR	VALEUR	Integer	5	0
4	abc MUTNUM	MUTNUM	String	8	0
5	abc MUTORI	MUTORI	String	8	0
6	DATEDT	DATEDT	Date	10	0
7	abc PROVENANCE	PROVENANCE	String	25	0
8	1.2 Y	Y	Double	18	8
9	1.2 X	X	Double	18	8
10	1.2 Z	Z	Double	18	8
11	1.2 PRECPLAN	PRECPLAN	Double	18	8
12	1.2 PRECALT	PRECALT	Double	18	8

Figure 43 : Exemple d'utilisation de l'outil de refactorisation sur la couche des points fixes (SITG).

2. Projets annexes QGis

D'autres traitements géomatiques, développements, personnalisation d'outils et projets ont été réalisés sur QGis. Certains sont en lien avec le projet principal présenté précédemment, et d'autres en sont indépendants. Parmi ces projets, quelques-uns ont été développés sur plusieurs SIG, en fonction des fonctionnalités présentes dans ces logiciels, certaines étant plus intuitives sur QGis ou sur ArcMap.

2.1 Dalles orthophotos

2.1.1 But du projet

L'objectif est de pouvoir créer une action qui permette de charger une ou plusieurs tuiles orthophotos en fonction de l'endroit cliqué sur la carte sur QGis. Les orthophotos du canton de Genève ont une résolution de 5 cm et sont sous forme de tuiles au format tif. Il y a un total de 436 tuiles, ce qui ralentit passablement le projet QGis qui les contient. Par conséquent, nous avons décidé de ne pas les charger dans le projet, mais de créer une action qui charge les orthophotos désirées manuellement.

2.1.2 Traitements

Nous avons à disposition les tuiles des orthophotos 2019, 436 au total, sous forme de TIFF. Dans un premier temps, nous les chargeons dans un projet QGis, par le biais d'un ajout de couche raster. Le but est de créer un « Index des tuiles », dans le Menu -> Raster -> Divers. Il faut sélectionner les orthophotos désirées, puis donner un nom au champ où enregistrer le chemin des fichiers des rasters indexés (Figure 44).

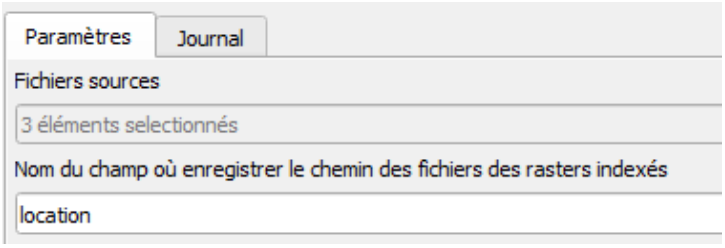


Figure 44 : Outil d'index des tuiles pour transformer les tuiles raster en un vecteur sous forme de grille.

En exécutant ce processus, un shapefile va se créer, qui sera une grille contenant les différentes tuiles, ayant la forme du canton de Genève (Figure 45). Dans la table attributaire de ce shapefile, il y a le chemin de chacune des tuiles sous le nom du champ attribué auparavant (« location »). C'est à ce champ que l'action pourra se référer par la suite. Ultérieurement, nous retirerons la symbologie et les étiquettes de ce shapefile, pour que la grille ne se superpose pas à des informations que nous souhaitons mettre en avant.

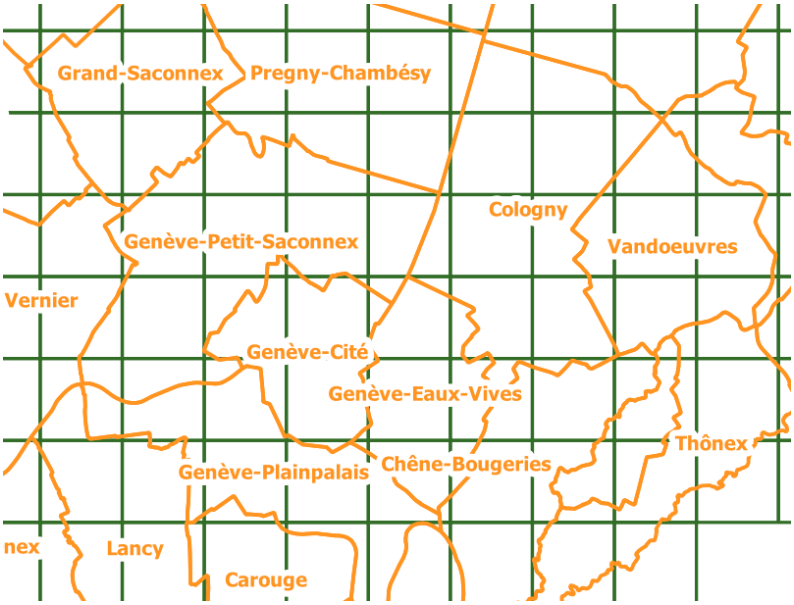


Figure 45 : Grille du shapefile créé à partir des tuiles raster (tif) des orthophotos 2019.

2.1.3 Action sur QGis

Dans les propriétés de la couche fraîchement créée, il suffit d'ajouter une action au shapefile, de type Python (Figure 46).

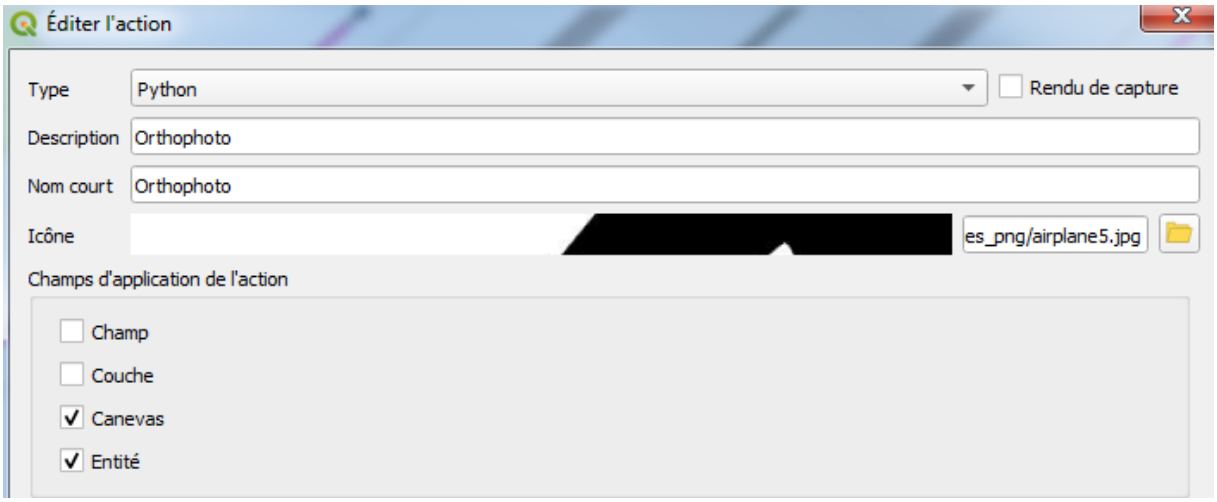


Figure 46 : Action de type Python implémentée sur la couche shapefile de l'index des orthophotos, permettant de charger l'orthophoto de la zone cliquée.

Le code Python permettant de créer l'action est le suivant :

```
import os # On charge le paquet os.
import ntpath # On importe le module pathname (WindowsNT/95 version).
def getVectorLayerByName(NomCouche):
    layermap=QgsProject.instance().mapLayers()# Il faut que la couche se trouve dans l'arborescence du projet ouvert.
    for name, layer in layermap.items():
        if layer.name()==NomCouche:
            if layer.isValid(): # On teste la validité de la couche.
               return layer
            else:
```

```
                return None
mypath=r"[% "location" %]" # On indique le chemin des tuiles raster.
instRegistry = QgsProject.instance()
nom=ntpath.basename(mypath) # Crée le lien avec le chemin indiqué auparavant (basename : renvoie la composante finale d'un nom de chemin).
raster_ouvert=getVectorLayerByName(nom) # On charge la tuile raster de l'endroit cliqué sur la carte.
if raster_ouvert is not None:
    QgsProject.instance().removeMapLayer(raster_ouvert.id())
    qgis.utils.iface.mapCanvas().refresh()
else:
    raster_ouvert = QgsRasterLayer(mypath,nom)
    QgsProject.instance().addMapLayer(raster_ouvert, False)
    root = QgsProject.instance().layerTreeRoot() # On charge la tuile raster dans l'arborescence des couches.
    g = root.findGroup("ORTHOPHOTOS") # On décide de placer la tuile dans un groupe défini dans le projet.
    g.insertChildNode(0, QgsLayerTreeLayer(raster_ouvert))
```

Les quatre dernières lignes de code permettent de choisir l'emplacement des orthophotos, une fois qu'elles sont chargées dans l'arborescence des couches. Ici, nous avons décidé de les mettre dans le groupe « ORTHOPHOTOS ».

Il est important de rajouter le préfixe **r** avant le chemin de la couche (ici : "[% "location" %]"). Le préfixe r signifie « chaîne brute », et il est nécessaire de l'utiliser sur les systèmes d'exploitation Windows lorsque le chemin contient des barres obliques inverses, sinon, la barre oblique inverse est traitée comme un caractère de tabulation au lieu d'un séparateur de chemin (https://docs.python.org/fr/3/howto/regex.html).

Finalement, il suffit de cliquer sur le bouton d'action et de cliquer sur n'importe quel carré de la grille pour que l'orthophoto soit chargée. L'orthophoto va automatiquement se mettre dans le groupe « ORTHOPHOTOS » dans l'arborescence des couches, mais il est possible de la supprimer en recliquant dessus sur la carte, pour ne pas ralentir le projet QGis.

2.2 Projet routes cantonales

2.2.1 But du projet

Ce projet est indépendant du projet principal sur QGis, et son propos est de mettre en évidence toutes les routes cantonales genevoises, afin de mesurer leur longueur totale. Nous souhaitons également mettre en évidence les ponts, les tunnels et les autoroutes (routes nationales), afin de créer une carte thématique compréhensible et visuelle.

2.2.2 Marche à suivre

1. Il faut tout d'abord télécharger la couche GMO_GRAPHE_ROUTIER sur le SITG et la charger dans un nouveau projet QGis. Cette couche contient l'attribut « CLASSIFICATION », qui comporte les différents types de routes, en l'occurrence « Non renseigné », « Privé », « Réseau communal primaire », « Réseau communal secondaire », « Route cantonale » et « Route nationale ». Ce sont les deux derniers types qui nous intéressent.
2. Sur **QGis**, il faut effectuer une sélection par attributs dans la table attributaire, en triant l'attribut CLASSIFICATION par ordre alphabétique et en sélectionnant « Route cantonale » et

« Route nationale ». Nous aurions pu faire la même chose sur ArcMap, en faisant un *Select by attribute*.

3. Nous avons enregistré cette sélection sous forme de shapefile, et avons créé une symbologie différente pour nos deux types de routes (cantonales et nationales).
4. Puis, dans la table attributaire, nous avons ouvert la calculatrice de champ, et avons additionné les valeurs de l'attribut SHAPE_LEN, qui correspond à la longueur de chaque tronçon de route. Nous avons effectué ce calcul pour les deux types de classifications, par le biais de l'expression suivante : *sum("SHAPE_LEN", group_by:="CLASSIFICA").* Ce calcul nous renvoie le résultat de 261663.4076 m (soit 261.66 km) pour les routes cantonales, et de 84623.7170 m (84.62 km) pour les routes nationales.
5. Les objets « tunnel » et « pont » se trouvent dans le shapefile CAD_OBJETDIVERS. Comme avant, nous avons trié la table attributaire pour avoir tous les polygones de tunnels et de ponts à la suite, puis les avons sauvés sous deux shapefiles.
6. Puis nous avons créé un *buffer* de 100 mètres autour de la couche des routes, sous Vecteur -> Outils de géotraitement -> Tampon.
7. Finalement, sur **ArcMap**, nous avons fait une intersection entre le *buffer* et les ponts, puis la même chose entre le *buffer* et les tunnels. Ceci a pour but de ne garder que les ponts et les tunnels qui touchent ou qui sont proches des routes. Nous avons utilisé ArcMap pour cette étape car l'intersection sur QGis ne fonctionnait pas comme souhaité.

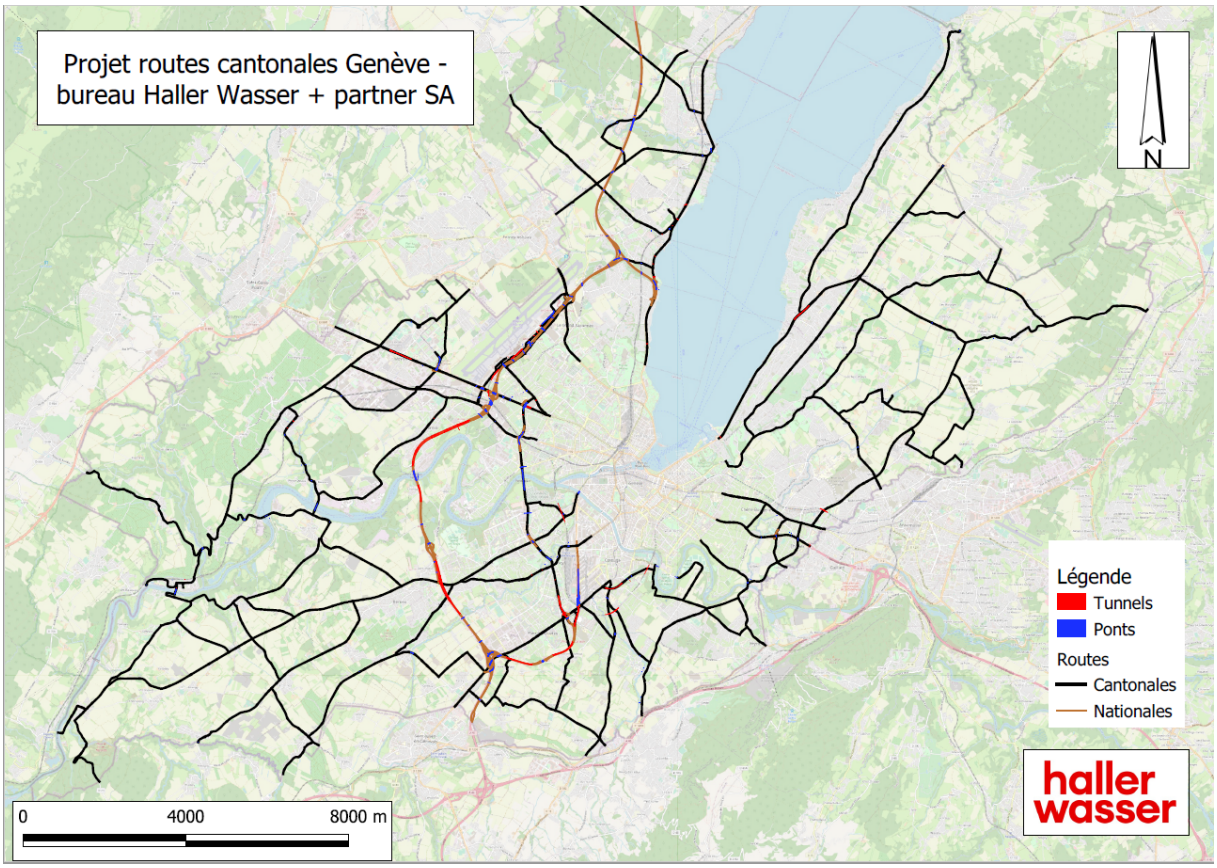


Figure 47 : Layout contenant les routes (cantonales et nationales) de Genève, les ponts, les tunnels et un fond de carte mondial.

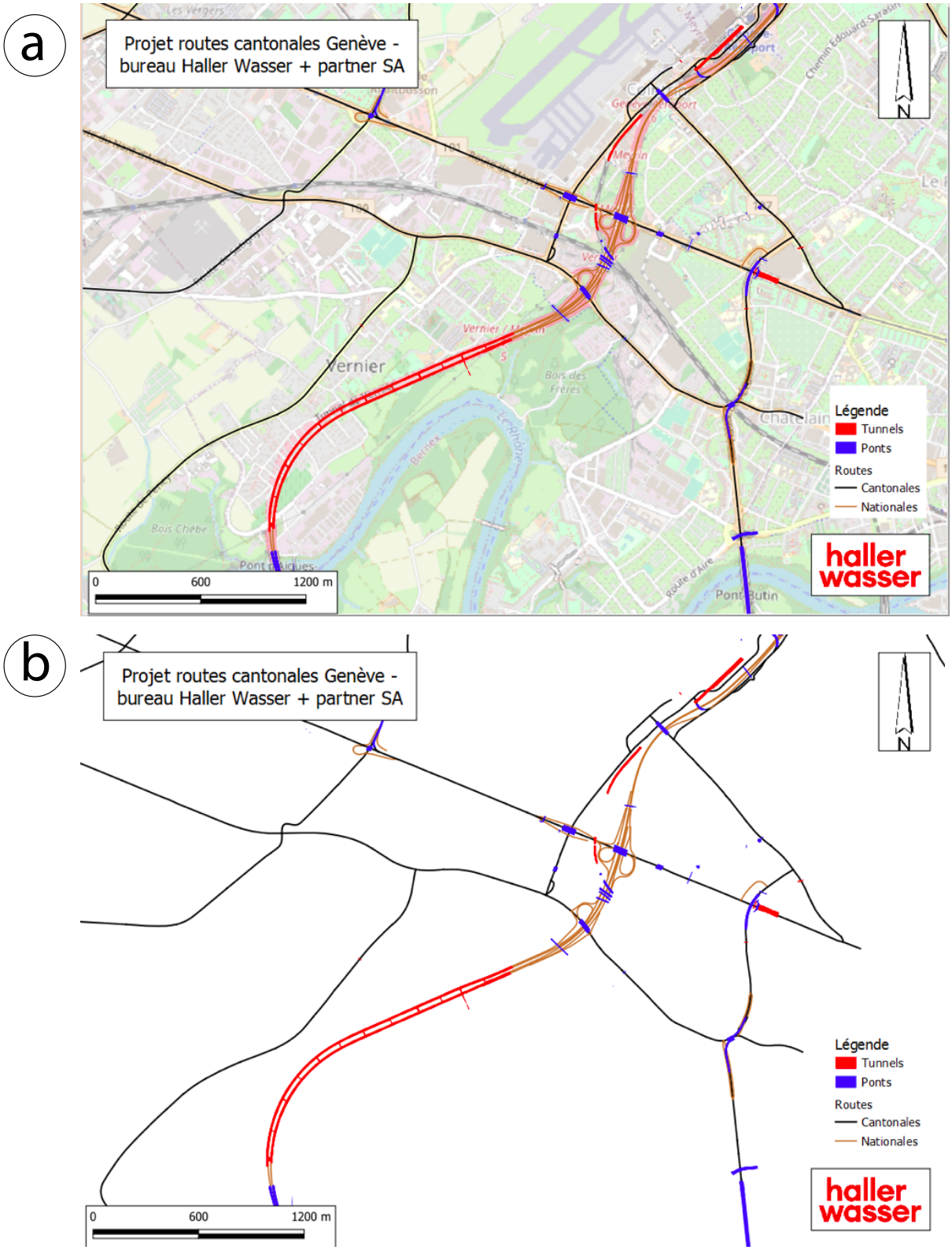


Figure 48 : Zoom sur le layout ; **a)** avec une carte de fond et **b)** sans carte de fond.

2.3 Etude de la voie verte Sécheron-Versoix

2.3.1 But du projet

Le but de ce projet est de tracer le parcours de la future voie verte qui reliera le Sécheron à Versoix. Cette voie est divisée en trois types de tronçons ; vert pour les zones mixtes à aménager, orange pour les sites propres à améliorer et rouge pour les sites propres à créer. Il est ensuite question de déterminer quelles sont les parcelles touchées par ce tracé, et de quel type il s'agit (privé ou public), pour ensuite évaluer un coût en fonction des divers tronçons.

2.3.2 Marche à suivre

Sur QGis

Nous disposons de zooms sur divers tronçons du tracé exact sous forme de pdf. Grâce à ce document et à l'aide d'un fond de carte mondial ainsi que des orthophotos 2019, nous pouvons nous repérer sur QGis afin de créer un tracé le plus précis possible.

En tout premier lieu, il faut créer une nouvelle couche shapefile  , et lui dessiner des entités ponctuelles sous forme de ligne  . Il faudra ensuite préciser l'id de la ligne dessinée (soit 1, 2 ou 3), pour leur attribuer des symbologies différentes par la suite. Pour rajouter une partie à une ligne déjà existante, il faut utiliser le bouton « ajouter une partie »  .

Une fois le tracé terminé (Figure 49a), il faut créer un *buffer* de 10m (5m de chaque côté de la ligne) autour du tracé, sous Vecteur -> Outils de géotraitement -> Tampon (Figure 49b).

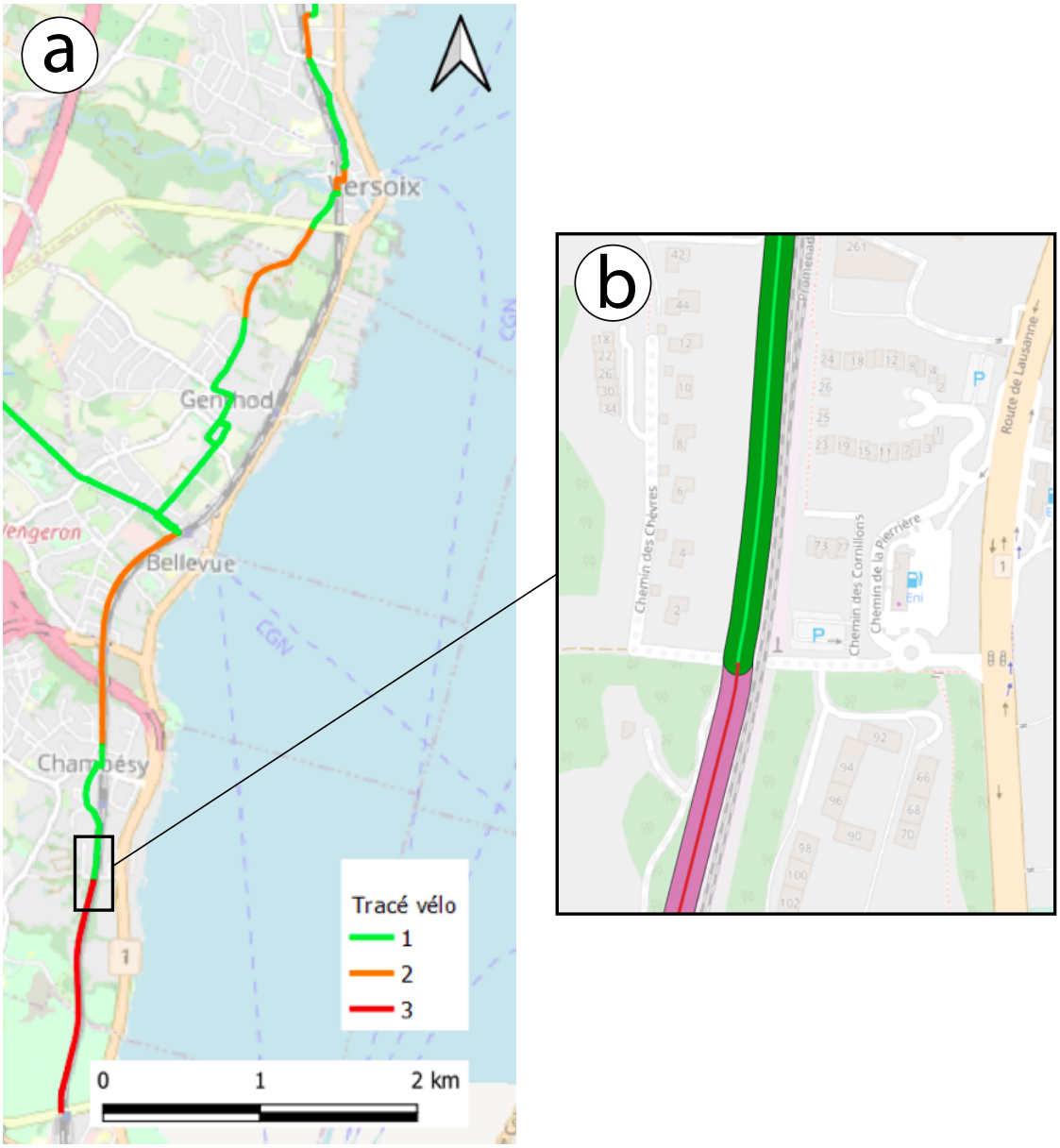


Figure 49 : **a)** Tracé de la voie verte avec les trois types de tronçons et **b)** zoom sur le buffer autour des tronçons 3 et 1, à proximité des voies ferrées.

Sur ArcMap

L'outil d'intersection ne fonctionnant pas sur QGis, nous sommes passés sur ArcMap, en utilisant l'outil *Intersection*. Nous avons d'abord fait une intersection entre la couche tampon et les parcelles, pour obtenir les parcelles impactées par ce tracé, dans le but de pouvoir déterminer si elles appartiennent au domaine public ou privé, afin d'évaluer un coût. Il est important de noter que dans le tronçon de type 3 (rouge), la majeure partie sont des parcelles appartenant aux CFF (privé).

Nous avons ensuite fait une intersection entre la couche tampon et les points limites, pour savoir lesquels seront touchés par ce projet, dans le but de les identifier et éventuellement les replacer.

Puis, nous sommes repassés sur QGis pour pouvoir exporter la table attributaire de ces deux couches d'intersection, afin d'obtenir un fichier Excel avec toutes les données, en complément aux cartes thématiques fournies.

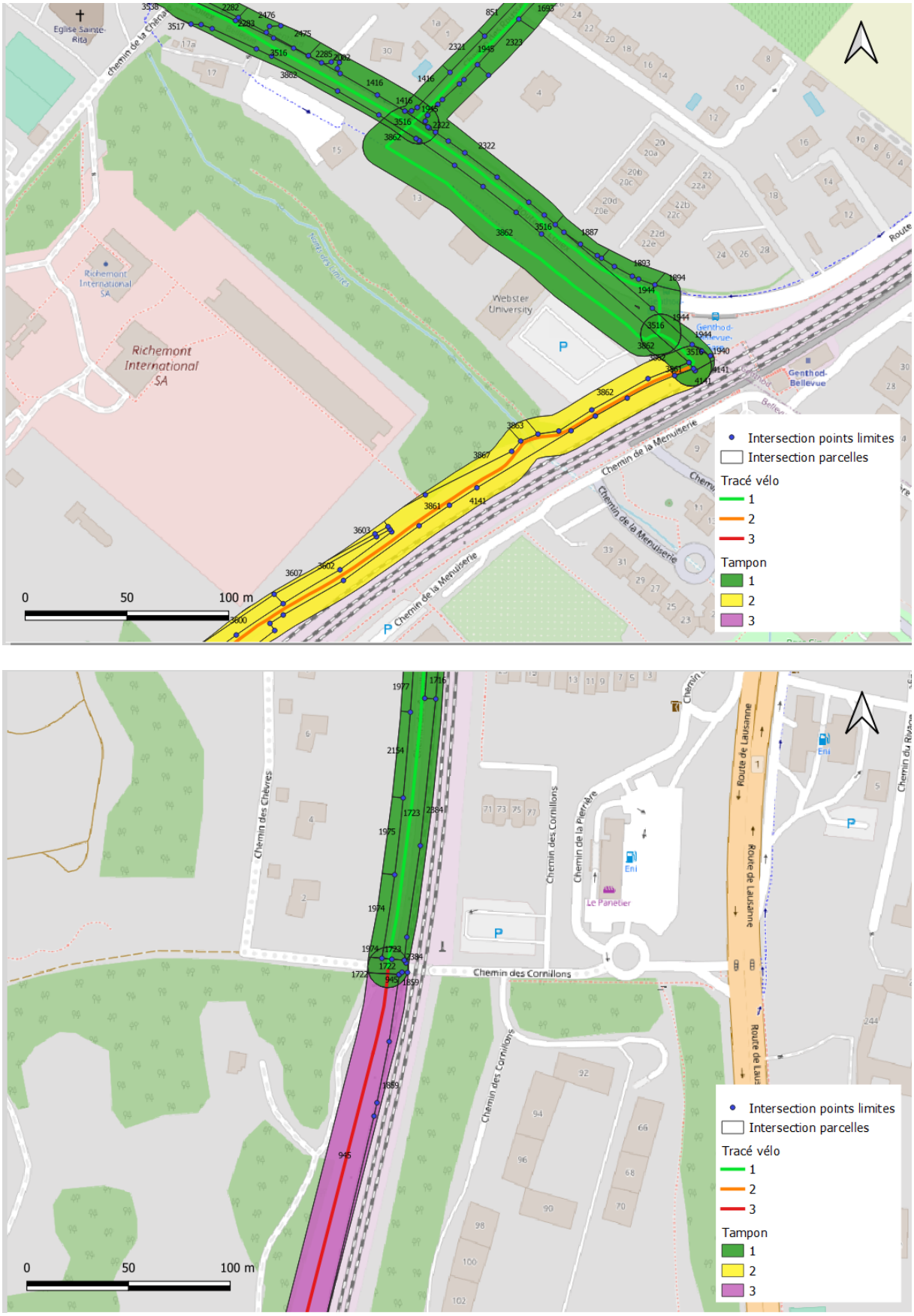


Figure 50 : Aperçu des deux types d'intersection : intersection avec les parcelles et avec les points limites impactés.

2.4 Projet points limites

2.4.1 But du projet

L'objectif est de mettre en évidence certaines communes genevoises en fonction des points limites de valeur 6 qu'elles contiennent. Ces points limites ne sont pas fiables et peuvent varier en quantité, c'est pourquoi il est important de les mettre en lumière, afin de décider lesquels fiabiliser ou non.

En fonction de la quantité de points contenus dans le plan RF d'une même commune, une symbologie différente va s'appliquer, et l'idée sera ensuite de créer des cartes thématiques mettant en évidence ces points, ainsi que d'autres couches pertinentes, afin d'obtenir une approche visuelle de la problématique.

2.4.2 Marche à suivre

L'idée est de créer plusieurs shapefiles en fonction des différentes communes d'intérêt (Carouge, Chêne-Bougeries, Cologny, Eaux-Vives, Grand-Saconnex, Lancy, Plainpalais, Petit-Saconnex et Vernier). Pour chaque commune, il faudra créer un shapefile contenant les points limites de valeur 6 et un shapefile des plans RF de la commune. Il faudra ensuite créer des intersections entre les communes d'intérêt et les couches des zones d'affectation et de bâtiments projetés, afin de restreindre l'information au niveau communal, et ne pas considérer l'entièreté de ces couches. Finalement, une analyse vecteur nous permettra de compter les points dans les polygones, en l'occurrence les points limites de valeur 6 dans les différents plans RF des communes d'intérêt.

1. Ajout des couches suivantes sur QGIS, provenant du SITG :

CAD_BATI_PROJET
SIT_AMENAG_ZONE
CAD_PLANRF
CAD_COMMUNE
CAD_POINT_LIMITE

2. Sélection d'entités par valeur  **+ sauvegarde des entités sélectionnées sous forme de shapefile :**

- Sélection par VALEUR de point (VALEUR=6) dans la couche point limite, et création d'un nouveau shapefile (point_limite_6).

- Sélection par COMMUNE dans la couche commune, pour avoir le contour des communes d'intérêt.

- Sélection par NO_COMM dans la couche point limite 6.

- Sélection par COMMUNE dans la couche plan RF, pour avoir le contour des plans des communes d'intérêt.

- Sélection par COMMUNE dans la couche SIT zones aménagement.

3. Intersection

- Entre la couche bâti projet et les communes d'intérêt (individuellement).

- Entre la couche SIT aménagement et les communes d'intérêt (individuellement).

4. Symbologies

Enregistrer les symbologies au format QML, afin de les appliquer à toutes les couches du même type.

5. Analyse vecteur

- Vecteur-> outils d'analyse-> compter les points dans les polygones, en fonction de la couche des plans RF (couche polygones) et des points limites 6 (couche points) de la commune d'intérêt (Figure 51).

- Créer une nouvelle couche « Count », qui aura une nouvelle colonne « NUMPOINTS » dans sa table attributaire, contenant le nombre calculé de points limites de valeur 6 par plan.

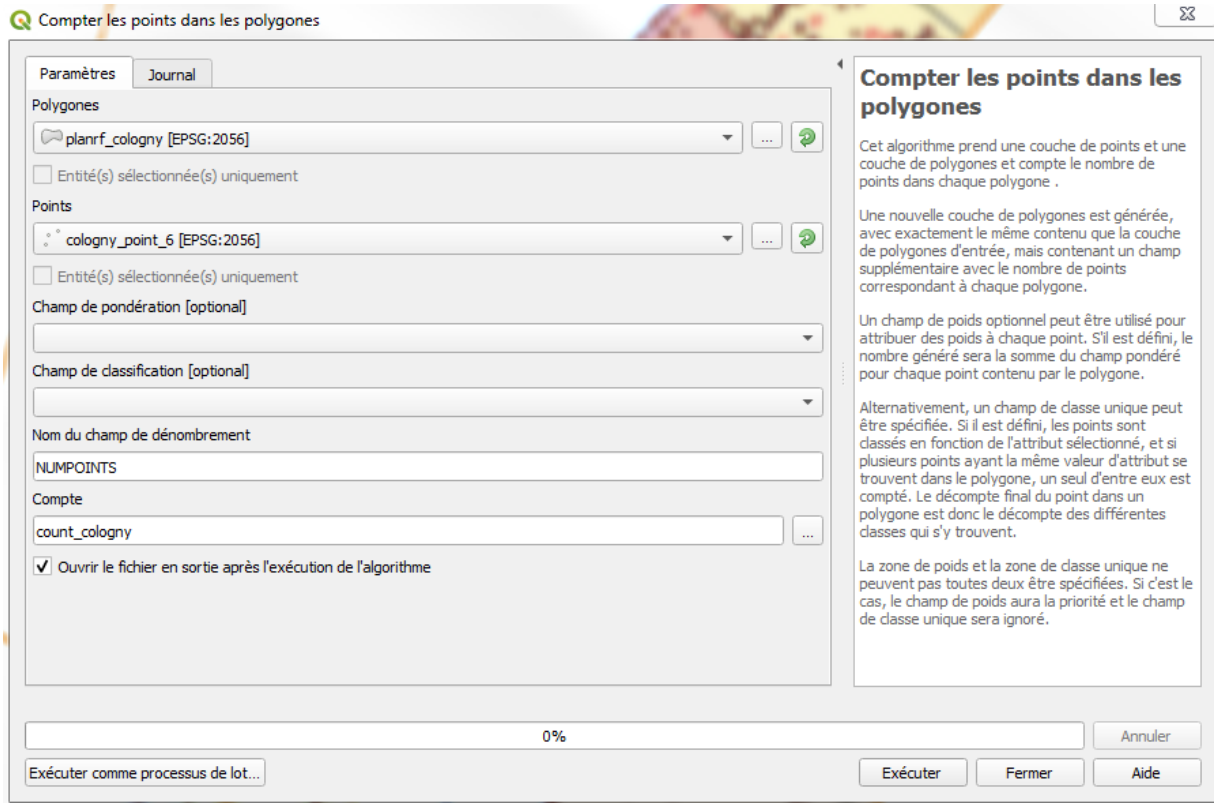


Figure 51 : Outil pour compter les points dans les polygones. Exemple de Cologny, avec la création d'une nouvelle couche (count_cologny), contenant l'attribut NUMPOINTS, qui est le total de points limites de valeur 6 comptés par plan RF dans la commune de Cologny.

6. Sélection par nombre de points dans un plan RF

Sélectionner les entités par valeur dans la couche « Count », où :

- NUMPOINT entre 0 et 50 : pas de symbologie

- NUMPOINT entre 50 et 100 : symbologie mauve transparent 40%

- NUMPOINT entre 100 et 200 : symbologie orange transparent 40%

- NUMPOINT > 200 : symbologie rouge transparent 40%

Pour chaque sélection : exporter-> sauvegarder les entités sélectionnées sous forme SHP.

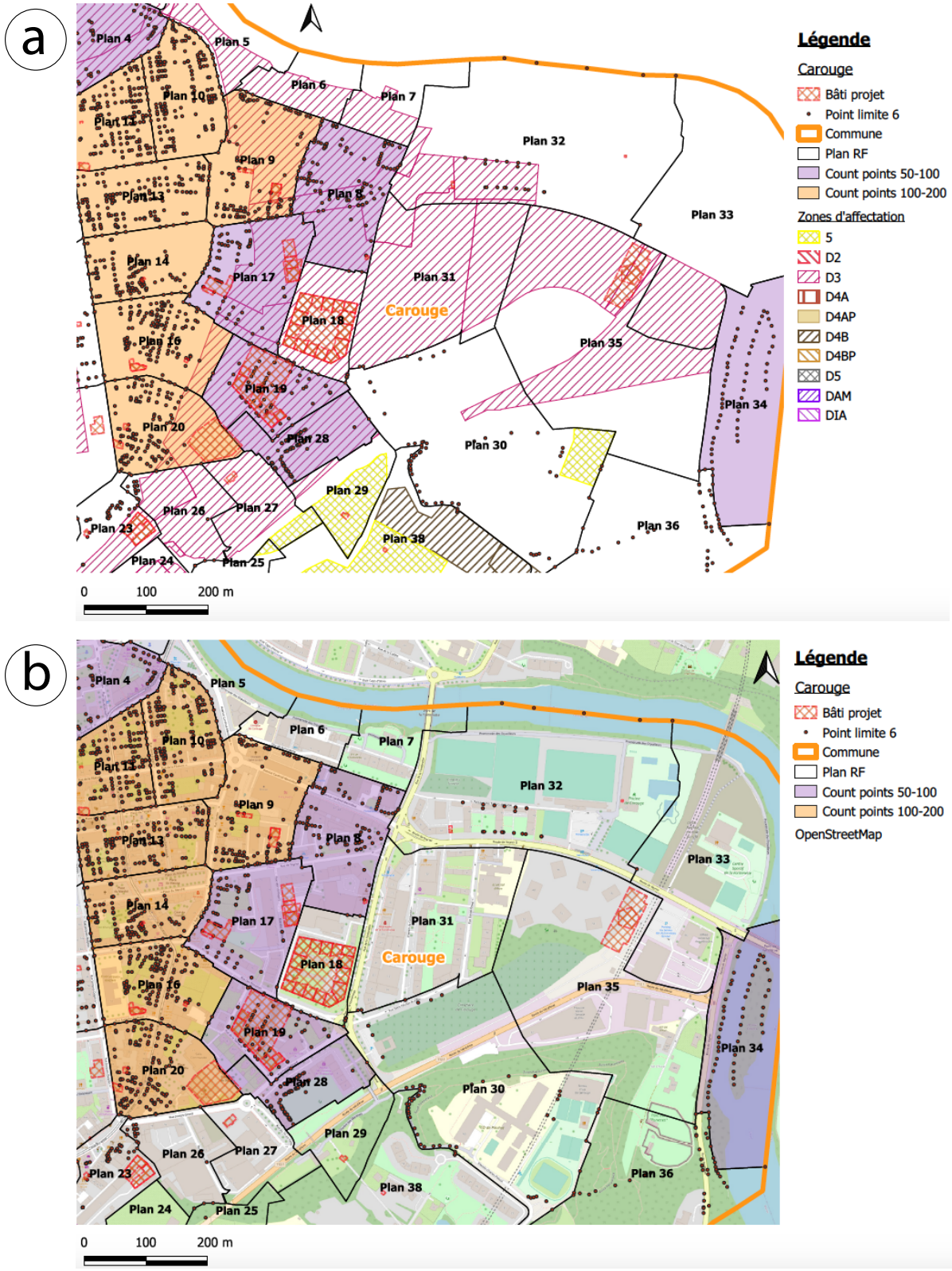


Figure 52 : Affichage de la zone de Carouge, avec **a)** les zones d'affectation et **b)** OpenStreetMap en fond de carte.

3. Projets ArcMap

Divers traitements géomatiques annexes sont réalisés sur ArcMap, soit pour compléter des développements de QGis, soit pour créer des symbologies prédéfinies sur certaines couches. Pour la création de ces symbologies, des étiquettes seront personnalisées via le constructeur d'expression. En parallèle, nous utiliserons l'outil d'étiquetage Maplex Label Engine, qui offre des paramétrages supplémentaires aux standards.

3.1 Projet symbologies

3.1.1 But du projet

Le propos est de créer des symbologies prédéfinies s'appliquant à certaines couches, pour pouvoir les charger ensuite dans n'importe quel projet par le biais de la barre d'outils TopoGeo, qui est un ESRI Add-in créé par Topomat technologies S.A. On ajoute cet Add-in sur ArcMap via Customize -> AddIn Manager. Trois barres d'outils vont se rajouter ; celle qui nous intéresse est la « TopoGeo Générale ». Il faut tout d'abord charger une géodatabase, contenant les couches nécessaires au projet, pour pouvoir lui appliquer les différentes symbologies créées.

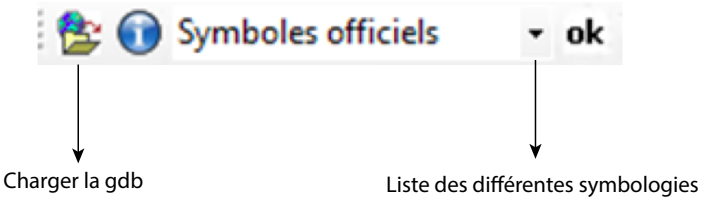


Figure 53 : Chargement d'une géodatabase via l'Add-in TopoGeo.

Il s'agit donc de personnaliser les symboles ainsi que les étiquettes des couches que nous souhaitons, et d'enregistrer le fichier sous forme de Layer File (lyr). Ce lyr sera ensuite intégré dans un fichier .xml, où il est possible de préciser si l'on veut qu'une couche soit visible (cochée ou décochée par défaut), sélectionnable (lorsque l'on est en mode édition) et quelle symbologie lui appliquer. Nous allons créer trois symbologies type ; une symbologie standard, une pour effectuer des contrôles et une pour l'extrait TM couleur.

3.1.2 Symbologies

Symbologie BASE

La symbologie standard reprend les lyr basiques qui se chargent automatiquement lorsque l'on ajoute une base de données sur ArcMap (gdb ou mdb) avec l'Add-in TopoGeo. Les symbologies de certaines couches ont cependant été adaptées, afin d'avoir une meilleure visibilité sur la carte. Des couches ont été mises en transparence ou avec une trame, pour voir à travers les couches qui se superposent.

Symbologie CONTROLES

Cette symbologie a pour but de faire des autos contrôles lorsque l'on modifie l'aire d'un bâtiment ou d'une parcelle. La Direction cantonale de la mensuration officielle gère deux types de surfaces : l'attribut SURFACE_TECHNIQUE et le SRFELE. La surface technique (similaire à l'attribut SHAPE_Area) est mise à jour à chaque modification de polygone. En revanche, le SRFELE, qui est une surface

arrondie au m^2, doit se modifier à la main. Le bouton SRF  dans la barre d'outils TopoGeo permet la mise à jour de cette surface (https://www.ge.ch/document/circulaires-dit-2004/annexe/8).

L'idée sera d'intégrer un calcul à l'étiquette, montrant la différence entre le SRFELE et le SHAPE_Area. Si la différence est supérieure à 0.5 m^2, cela signifie que le SRF n'a pas été mis à jour. Il est important de pouvoir faire ce contrôle avant d'envoyer des documents au cadastre, car le *Web Checker* ne décèle pas ce genre d'erreurs.

Symbologie Extrait TM couleur

Le registre foncier propose une variante en couleurs du plan du registre foncier, qui s'applique à quelques couches (Figure 54). Pour le reste, toutes les couches qui possèdent l'attribut MUTVERSION se verront affecter une symbologie en fonction de cet attribut, pour les cas où la MUTVERSION = 0, 10 et 90. Pour les versions 0 et 10, la symbologie est assez simpliste et de couleur noir/blanc. Lorsque la version est égale à 90, cela signifie qu'une entité de la couche a été modifiée, par conséquent la symbologie sera entourée d'un contour rouge, pour distinguer les entités modifiées ou les nouvelles entités des entités inchangées. Toutes les couches possédant un attribut mutversion peuvent être modifiées, par exemple les bâtiments, les façades, les escaliers, les murs, les parcelles etc.

Signe conventionnel	Désignation
	Bâtiment (bâtiment projeté)
	Bâtiment souterrain & réservoir (éléments projetés)
	Eau stagnante Eau courante Bassin
	Route - chemin
	Autre surface en revêtement dur Trottoir, place d'aviation
	Forêt dense

Figure 54 : Couleurs à appliquer pour la variante en couleurs du plan du registre foncier.

3.1.3 Exemple de code du fichier XML

```
<!-- TEST VB  -->
<Filter type="1" alias="TEST VB"> # Nom de la symbologie.
  <Layers> # Couches pour lesquelles on souhaite appliquer une symbologie prédéfinie (fichier .lyr) et que l'on souhaite cocher/décocher par défaut dans le projet lorsque l'on charge la symbologie.
   <Layer featureclass="CAD_BATIMENT_HORS_SOL" file="attr_cad_bati_hs.lyr" visible="true" selectable="false" /> # Visible = true permet de cocher cette couche dans le projet. Selectable = false implique que la couche n'est pas sélectionnable, c'est-à-dire qu'il n'est pas possible d'utiliser les outils de sélection interactifs pour y sélectionner des entités (indiqué comme Not Selectable sous l'onglet  de la table des matières).
    <Layer featureclass="CAD_NATURE_SOL" file="attrib_cad_nature.lyr" visible="true" selectable="false" />
    <Layer featureclass="CAD_DOMROUTIER_SURFACES_NIV0" file="attrib_cad_niv0_domroutier.lyr" visible="true" selectable="false" />
    <Layer featureclass="CAD_DOMROUTIER_SURFACES_NIV1" file="attr_cad_niv1_domroutier.lyr" visible="false" selectable="false" /> # Visible = false permet de décocher par défaut cette couche dans le projet.
    <Layer featureclass="CAD_DOMROUTIER_SURFACES_NIV2" file="attr_cad_niv2_domroutier.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_DOMROUTIER_SURFACES_NIV-1" file="attr_cad_niv-1_domroutier.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_DOMROUTIER_SURFACES_NIV-2" file="attr_cad_niv-2_domroutier.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_BATIMENT_SOUS_SOL" file="attr_cad_bati_ss.lyr" visible="true" selectable="false" />
    <Layer featureclass="CAD_POINT_LIMITE" file="attrib_cad_point_limite.lyr" visible="true" selectable="false" />
    <Layer featureclass="CAD_POINT_PARTICULIER" file="attrib_cad_point_situation.lyr" visible="true" selectable="false" />
    <Layer featureclass="CAD_ADRESSE" file="attr_cad_adresse.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_DDP" file="attr_cad_ddp.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_BIENS_FONDS" file="attr_cad_biens_fonds.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_PLAN" file="attr_cad_plan_rf.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_OBJDS_MUR" file="attri_cad_objds_mur.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_OBJDS_ESCALIER_RAMPE" file="attr_cad_objds_escalier_rampe.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_OBJDS_AUTRE_CORPS_BATI" file="attr_cad_objds_autre_corps_batiment.lyr" visible="false" selectable="false" />
    <Layer featureclass="CAD_OBJDL_MUR_MITOYEN" file="attr_cad_objdl_mur_mitoyen.lyr" visible="false" selectable="false" />
    <Default visible="false" selectable="false" /> # Default visible = false implique que toutes les autres couches qui n'ont pas été associées à une symbologie seront décochées par défaut.
  </Layers>
 </Filter>
```

3.1.4 Etiquettes

Sur ArcMap, il est possible de créer des étiquettes personnalisées par le biais d'une expression, en utilisant soit le langage JScript, VBScript ou Python. Dans notre cas de figure, il est utile de pouvoir modifier l'étiquette en fonction de plusieurs paramètres :

- Choisir l'ordre des informations (avec ou sans sauts de ligne).

- Rajouter du texte et/ou des parenthèses (en plus des attributs) dans l'étiquette.

- Donner une abréviation à des entités d'attributs.

- Insérer un calcul dans l'étiquette.

Tous ces paramètres se gèrent dans l'onglet *Labels*, dans les propriétés de la couche.

3.1.4a Ordre des informations

En VBScript, si l'on souhaite montrer des attributs dans un ordre bien particulier, il faut mettre les noms des champs dans l'ordre souhaité, et les séparer par des guillemets et des esperluettes. Si l'on veut rajouter des sauts de ligne, il faut insérer *& vbCrLf &*.

Exemple de code pour le domaine routier : *[OBJET] & " " & vbCrLf & [REVETEMENT]*

3.1.4b Texte et parenthèses dans l'étiquette

Exemple de code pour le domaine routier, contenant du texte en plus des attributs :

[OBJET] & " " & "(Niv: " & [NIVEAU] & ")" & vbCrLf & [REVETEMENT]

3.1.4c Abréviation d'attribut

Pour créer des abréviations, il faut activer le mode *Advanced* dans le constructeur d'expressions sous l'onglet *Labels*, et choisir un type de langage. Il va falloir ensuite créer une fonction qui va nous permettre de choisir dans quel attribut l'on souhaite faire des abréviations, et ce que l'on souhaite remplacer par quoi. Il est possible d'utiliser les trois types de langages à disposition, *i.e.* Python, JScript et VBScript. Dans les exemples ci-dessous, nous allons voir la différence entre le JScript et VBScript.

Par ailleurs, il est possible de choisir entre deux méthodes ; soit étiqueter toutes les entités de la même façon, soit créer des classes d'entités et les étiqueter différemment (Figure 55).

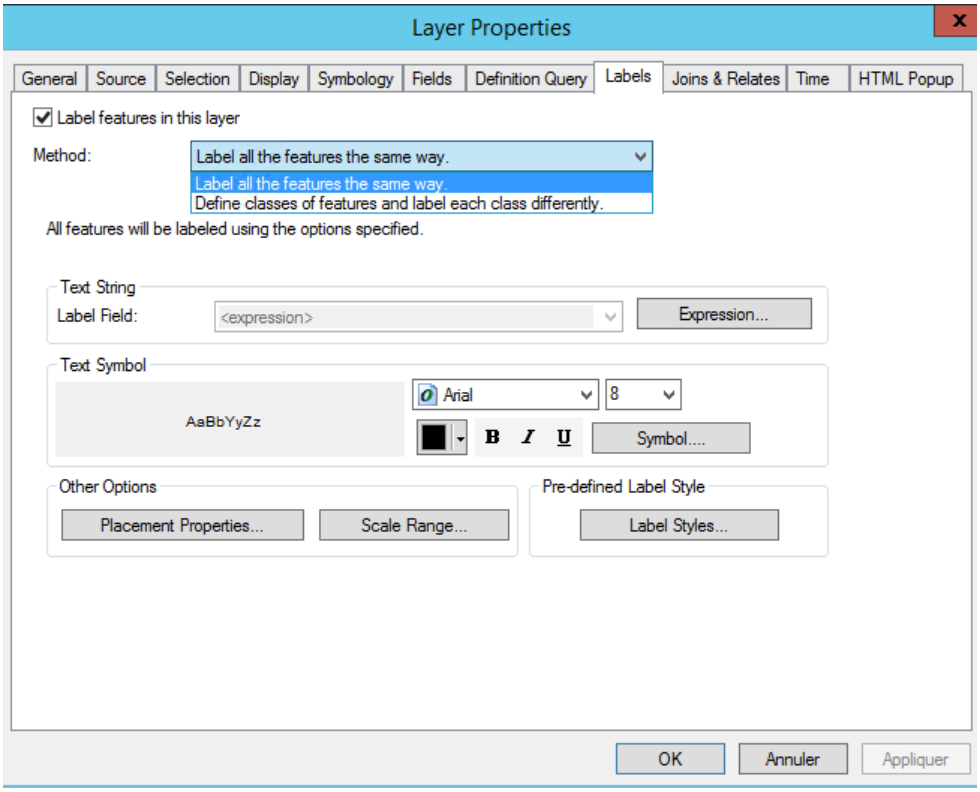


Figure 55 : Choix entre les deux méthodes d'étiquetage, sous l'onglet Labels dans les propriétés de la couche.

- Plusieurs classes : Il est possible de créer une ou plusieurs classes, en fonction de l'utilisation de notre étiquette. Si, en fonction de la valeur de tel ou tel champ, on souhaite afficher un certain attribut, il faudra créer autant de classes que de possibilités. Par exemple, en fonction de la valeur du numéro de point de situation, on veut afficher un certain attribut : si le point <= 0, on affiche le numéro provisoire (NO_PROV), en revanche, s'il est >0, on affiche le numéro de point (NO_POINT). De plus, cela permet de personnaliser la police, la taille et la couleur de l'étiquette en fonction de la classe.

Exemple 1 : Sur la couche « cad_objds_piscine », nous avons créé deux classes, une pour le type privé et une pour le public, car c'est les seuls deux cas de figures existants dans l'attribut « TYPE ». Puis, pour chacune des deux classes, nous avons écrit le bout de code suivant en JScript dans le mode *Advanced*, pour abréger le type de possesseur d'une piscine, *i.e.* soit privé (« ~Pr »), soit public (« ~Pu »).

1. Classe pour privé

```
function FindLabel ( [TYPE]  ) # On précise que la fonction s'applique à l'attribut TYPE.
{string= [TYPE] # On précise dans quel attribut on a du texte à modifier (ici, il s'agit de TYPE à nouveau).
newstring=string.replace('Privé', '~ Pr') # On remplace l'ancien texte (Privé) par le nouveau (~ Pr).
return newstring # On ne garde que le nouveau texte.
  return ;
}
```

2. Classe pour public

```
function FindLabel ( [TYPE]  )
{string= [TYPE]
newstring=string.replace('Public', '~ Pu')
```

```
return newstring
 return ;
}
```

En revanche, lorsque l'on souhaite créer des étiquettes un peu plus spécifiques, contenant plusieurs attributs et donc plus de possibilités, il est préférable de ne créer qu'une seule classe et de faire une fonction plus complète. Autrement, les étiquettes seront certes correctes, mais vont se placer aléatoirement dans le polygone (ou tout autre type de géométrie), et l'étiquette ne contiendra jamais toutes les informations au même endroit.

- Pas de classes : permet d'écrire une fonction un peu plus complexe, qui s'applique à plusieurs attributs de la couche.

Exemple 2 : Sur la couche « cad_niv_domroutier », on souhaite créer une étiquette contenant l'objet, le revêtement et le niveau. On souhaite également abréger les entités des attributs OBJET et REVETEMENT. Ici, il faudra donc créer deux fonctions d'abréviation (nommées « abrev ») pour les deux attributs, et une fonction qui permet de les concaténer (fonction « FindLabel »). Nous activons à nouveau le mode *Advanced* et codons en VBScript.

```
Function abrev_objet ( [OBJET]) # On crée notre fonction abrev_objet qui s'applique à l'attribut OBJET.
 if ( [OBJET]  = "Chaussée") then # On indique quelle entité va être modifiée dans l'attribut OBJET.
 abrev_objet = "c" # On note l'abréviation souhaitée pour l'entité précédente.
 elseif ( [OBJET]  = "Chemin") then
 abrev_objet = "jd"
 elseif ( [OBJET]  = "Espace de stationnement") then
 abrev_objet = "e"
 elseif ( [OBJET]  = "Ilot circulation") then
 abrev_objet = "i"
 elseif ( [OBJET]  = "Ilot latéral") then
 abrev_objet = "il"
 elseif ( [OBJET]  = "Parking") then
 abrev_objet = "p"
 elseif ( [OBJET]  = "Piste cyclable") then
 abrev_objet = "pc"
 elseif ( [OBJET]  = "Place d'aviation") then
 abrev_objet = "pav"
 elseif ( [OBJET]  = "Site propre TC") then
 abrev_objet = "tc"
 elseif ( [OBJET]  = "Surface latérale") then
 abrev_objet = "sl"
 elseif ( [OBJET]  = "Trottoir") then
 abrev_objet = "tr"
 end if
End Function
```

```
Function abrev_revet ( [REVETEMENT] ) # On crée notre fonction abrev_revet qui s'applique à l'attribut REVETEMENT.
 if ( [REVETEMENT] = "Arbustes") then
 abrev_revet = "a"
 elseif ( [REVETEMENT] = "Béton") then
 abrev_revet = "b"
 elseif ( [REVETEMENT] = "Béton bitumineux") then
 abrev_revet = "bb"
```

```
 elseif ( [REVETEMENT] = "Gazon") then
 abrev_revet = "g"
 elseif ( [REVETEMENT] = "Grille gazon") then
 abrev_revet = "gg"
 elseif ( [REVETEMENT] = "Graviers") then
 abrev_revet= "gr"
 elseif ( [REVETEMENT] = "Prairie") then
 abrev_revet = "pr"
 elseif ( [REVETEMENT] = "Plates-bandes") then
 abrev_revet = "pb"
 elseif ( [REVETEMENT] = "Pavés") then
 abrev_revet = "pa"
 elseif ( [REVETEMENT] = "Terre") then
 abrev_revet = "t"
 elseif ( [REVETEMENT] = "Autre") then
 abrev_revet = "au"
 end if
End Function
```

```
Function FindLabel ( [OBJET] , [NIVEAU] , [REVETEMENT] ) # On crée une fonction qui sera la concaténation des attributs que l'on souhaite afficher.
 FindLabel = abrev_objet ([OBJET]) + " - " + abrev_revet ( [REVETEMENT] ) & vbCrLf & + " [Niv= " + [NIVEAU] + "]" # On crée la suite d'attributs qui nous intéresse, en appelant les deux fonctions créées auparavant (« abrev »). Par ailleurs, on souhaite que l'objet et le revêtement soient séparés d'un tiret, et que le niveau apparaisse à la ligne.
End Function
```

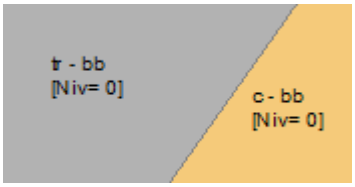


Figure 56 : Exemple d'étiquette avec l'abréviation de l'objet et du revêtement, puis le niveau à la ligne.

Le code contenant les étiquettes personnalisées des couches CAD_NATURE_SOL et CAD_POINT_LIMITE se trouve dans l'annexe A1.

3.1.4d Calcul dans l'étiquette

Il est possible de faire des calculs basiques entre les attributs, tels que des soustractions, additions, multiplications et divisions. D'autres fonctions peuvent également être insérées dans le calcul, notamment faire des arrondis (*round*).

Exemple de soustraction et d'arrondi pour la couche des parcelles, en VBScript: *[NOPARC] & " (" & [GENRE] & ")" & vbCrLf & [SRFELE] & " m2 [" & round ([SRFELE] - [SHAPE_Area] ,2) & "]"*

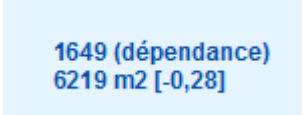


Figure 57 : Exemple d'étiquette de contrôle pour les parcelles, avec le numéro de parcelle, le genre, le SRFELE et la différence entre le SRFELE et la surface (arrondi à deux décimales).

Exemple de soustraction et d'arrondi pour la couche des bâtiments : *[NOBAT] & " (" & [MUTVERSION] & ")" & vbCrLf &[CODNOM] & vbCrLf & [SRFELE] & " m2 [" & round([SRFELE] - [SHAPE_Area] ,2) & "]" & vbCrLf & "nhs = " & [NIVEAU_HORSOL] & " - nss = " & [NIVEAU_SSOL]& vbCrLf & "Prov. = " & [PROVENANCE] & vbCrLf & "N° autor = " & [NO_AUTOR]*

Cette étiquette s'applique aux bâtiments dont la surface a été modifiée. Si la MUTVERSION est de 0, on n'affiche que le numéro du bâtiment. En revanche, si elle est égale à 10 ou 90, on affiche une étiquette de contrôle. L'aire se modifie automatiquement (155 m^2), en revanche, si l'on ne remet pas à jour le SRFELE, la valeur entre crochet sera supérieure à 0.5 m^2 (qui est la tolérance de l'arrondi), ce qui est erroné. Dans la figure ci-dessous, on remarque que pour la MUTVERSION 10, l'écart est correct, alors que pour la MUTVERSION 90, l'écart est trop élevé, il faudra par conséquent mettre à jour le SRFELE (Figure 58).

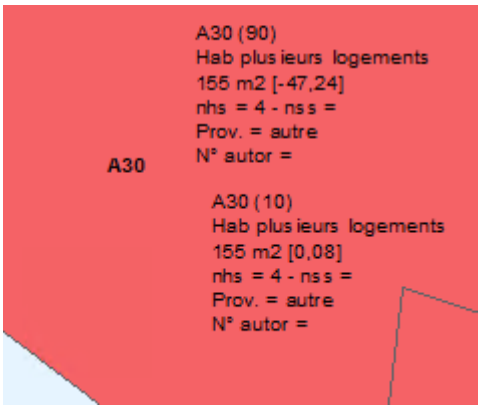


Figure 58 : Exemple d'étiquette de contrôle pour les bâtiments, avec les trois types de MUTVERSION (0, 10 et 90).

3.1.5 Maplex Label Engine

En parallèle aux systèmes d'étiquetage standards, il est possible d'activer l'outil d'étiquetage Maplex, qui propose des outils permettant d'améliorer les étiquettes de la carte, afin de contrôler le positionnement, la taille, les abréviations etc. des couches. Il faut tout d'abord mettre la barre d'outils *Labeling*, et cocher *Use Maplex Label Engine* pour l'activer. Ou bien aller dans les propriétés du groupe de couches, sous l'onglet général, choisir *Maplex Label Engine* dans *Label Engine*.

3.1.5a Création d'abréviations

Dans la barre d'outils *Labeling*, aller dans *Abbreviation Dictionaries* (Figure 59).

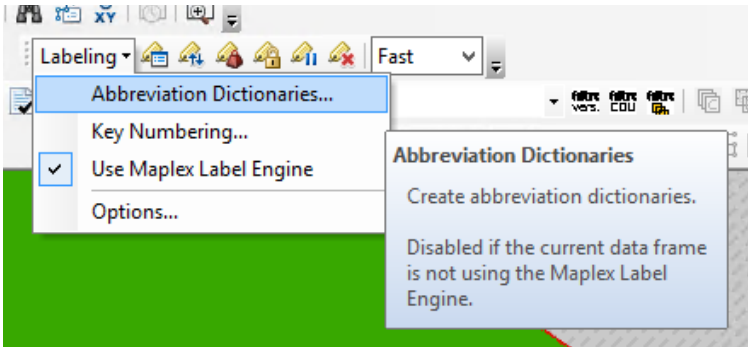


Figure 59 : Barre d'outils Labeling pour la gestion des étiquettes dans le système d'étiquetage Maplex.

Il faut ensuite créer un nouveau dictionnaire, en ajoutant les abréviations souhaitées. Dans l'exemple ci-dessous, nous avons abrégé les types d'adresses, pour réduire la longueur du texte de l'étiquette.

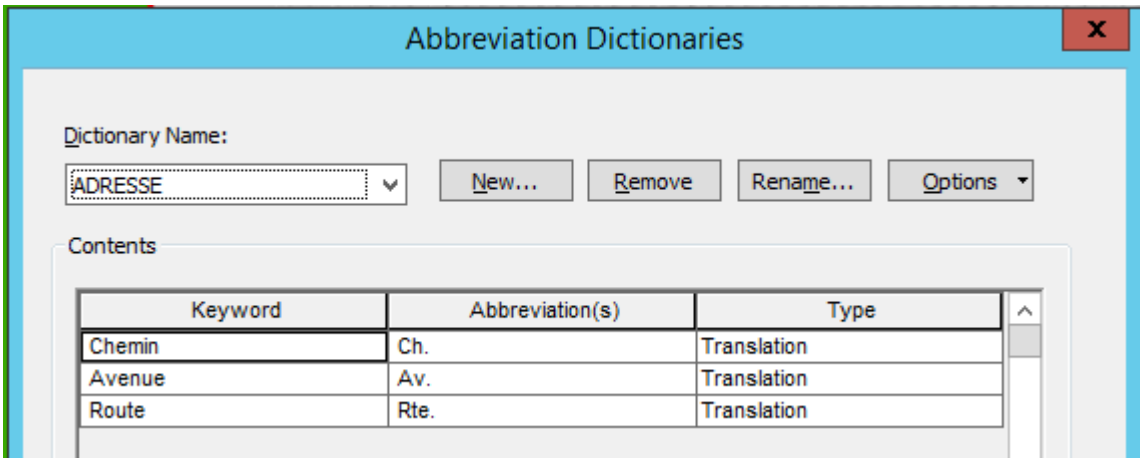


Figure 60 : Exemple de création de dictionnaire pour la couche des adresses (Maplex).

Ensuite, cliquer sur le bouton *Label Manager* (Figure 61), pour pouvoir gérer les paramètres des couches. C'est ici que l'on peut gérer le positionnement, l'activation des abréviations, l'angle des étiquettes, la résolution de conflit avec d'autres étiquettes etc.

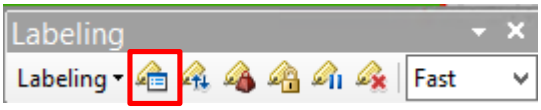


Figure 61 : Bouton Label Manager, permettant de gérer les différents paramètres d'étiquettes.

3.1.5b Inconvénients de Maplex

Maplex n'est pas installé par défaut sur ArcMap, il faut donc l'activer à chaque fois, recharger les dictionnaires qu'on a créés et les appliquer aux couches qui nous intéressent. Cela peut vite devenir fastidieux si l'on a défini des paramètres sur plusieurs couches, car il faudra faire plusieurs fois les mêmes manipulations.

3.2 Projet canevas

3.2.1 But du projet

L'idée est de créer une symbologie et des étiquettes types pour les canevas. Le canevas topographique se compose d'un ensemble de points, dont la position est connue avec une très grande précision et il sert de référence aux géomètres pour les relevés de terrain (www.colombes.fr). Lors d'un relevé, les géomètres se mettent en station avec le théodolite (instrument servant à la mesure de directions), en positionnant l'instrument de manière à ce que son axe vertical soit perpendiculaire au plan horizontal de la station (Wikipedia). Cela permet de déterminer les coordonnées des points d'appui du canevas, à partir d'un point du réseau de référence. Ils créent ainsi un réseau de points et de visées, qui seront ensuite téléchargés sur ArcMap ou AutoCAD, pour visualiser le relevé effectué sur le terrain.

3.2.2 Marche à suivre

Nous avons à disposition deux types de shapefiles ; « réseau_point » et « réseau_visée », et l'on souhaite créer deux fichiers lyr par shapefile. Il y aura un fichier lyr général, et un plus spécifique, contenant plus de détails.

3.2.3 Données

Points

- **Réseau de cibles** : 9100-9500. Les cibles sont des objets, réfléchissants ou non, généralement situés en hauteur, faisant office de point de référence pour une mise en station.

- **Nouvelle station** : 9600.

- **Points de détail** : 8000. Il s'agit des points acquis lors d'un levé de détails, qui est une opération topographique consistant à déterminer la position (en XYZ) des différents objets, artificiels ou naturels (angle de bâtiment, porte, détail de trottoir, candélabre, tampon, crête de talus, fossé, etc.) existants sur le terrain, à partir des points de canevas. En général, la méthode employée est le rayonnement à partir d'une ligne d'opération orientée (www.biblio.univ-antananarivo.mg).

- **Point fixes (PF)** : 100-7999. Ce sont tous les points matérialisés de manière pérenne (chevilles, bornes, etc.), placés très précisément sur le terrain et permettant aux géomètres de faire des mesures. On distingue deux types de points fixes : les points fixes planimétriques (PFP) et les points fixes altimétriques (PFA) (www.vd.ch).

Attributs

Cat XY
0 : Point fixe
1 : Compensée
2 : Approchée
3 : A calculer
4 : Inexistante

Station
0 : Pas de station (pas de symbole)
1 : Station (symbole)

3.2.4 Fichiers lyr

Couche réseau_point

Symbologie générale

Concernant la couche « réseau_point », nous souhaitons créer une symbologie complète, contenant les stations, les points de détail et les points fixes. Nous voulons afficher des informations différentes selon les attributs, il faut par conséquent définir des classes d'entités dans le groupe des labels (Figure 62).

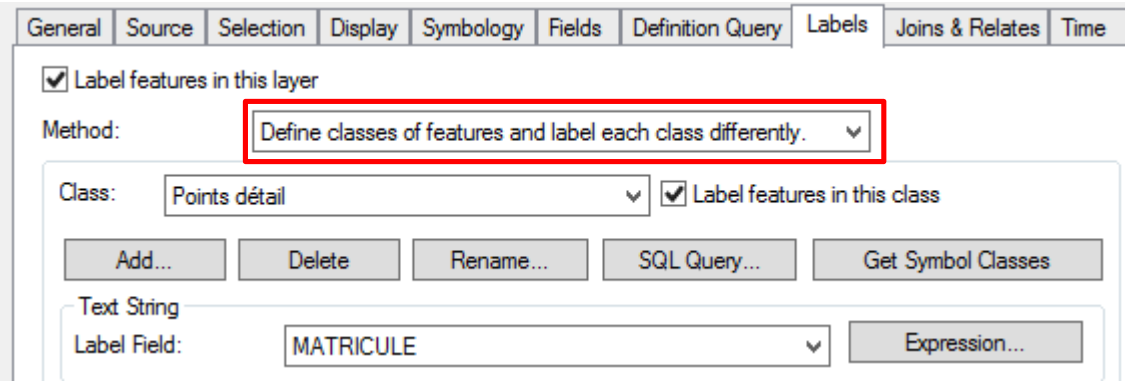


Figure 62 : Définition de classes pour créer différents types de labels.

Les points fixes et les nouvelles stations doivent avoir un label contenant le numéro de commune, le numéro de plan et le numéro de matricule. En tout premier lieu, on impose une requête SQL pour déterminer quels types d'entités vont être touchés par les étiquettes, en l'occurrence les nouvelles stations (station = 1) et les points fixes (station = 0 et catxy =0).

SQL Query: "STATION" =1 OR ("STATION" =0 AND "CATXY"=0)

Puis, dans le constructeur d'expressions, on écrit en langage JScript :

[Commune] + " " + [Plan] + " " + [MATRICULE]. *Exemple : 24 075 117*. **NB** : par convention, les points fixes n'ont pas de plan (000) et les nouvelles stations ont un plan 0.

Concernant les points de détail, on souhaite afficher uniquement le matricule. Toutes les valeurs de l'attribut CATXY qui sont différentes de 0 font référence aux points de détail.

SQL Query: "STATION" =0 AND ("CATXY" =1 OR "CATXY" =2 OR "CATXY" =3 OR "CATXY" =4)

Et ensuite, dans le constructeur d'expression de l'étiquette on ajoute l'attribut matricule. Il est également possible de n'afficher qu'une certaine partie du matricule, par exemple, que les trois derniers chiffres (soit 023 plutôt que 8023). Pour ce faire, il faut écrire une expression (en VBScript par exemple) où l'on note : Right([MATRICULE], 3). Ou bien, on peut noter [MATRICULE] – 8000. Dans ce cas, le point 8023 devient le 23.

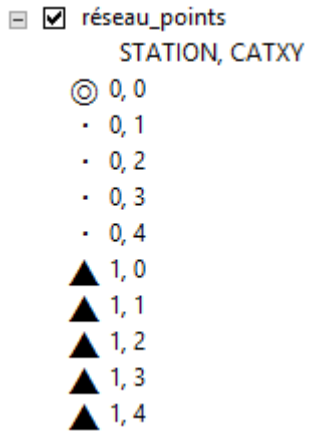


Figure 63 : Symbologie en fonction de la station et de la catxy (catégories : unique values, many fields). Ici, le double rond est un point fixe, les petits points noirs sont des points de détail et les triangles noirs sont des nouvelles stations.

Symbologie restreinte

L'autre symbologie est plus restreinte, elle ne contient que les stations et les points fixes. Pour ce faire, il faut faire une requête SQL dans les propriétés de la couche (*Definition query*), afin de réduire les possibilités des attributs souhaités dans la symbologie, les étiquettes et la table attributaire. Cela permet de ne garder que les informations que l'on juge pertinentes.

Definition query de la couche réseau_point : "STATION" =1 OR ("STATION" =0 AND "CATXY"=0)

Ensuite, dans l'expression du label, on note en JScript : [Commune] + " " + [Plan] + " " + [MATRICULE].

Figure 64 : Symbologie en fonction de la station et des points fixes (0 = pas de station (point fixe) et 1 = station) (catégories : unique values).

Couche réseau_visée

Symbologie générale

Concernant la couche « réseau_visée », la première symbologie englobe toutes les visées (stations, points de détail et points fixes). Les visées peuvent être de trois types : d'une station à l'autre, d'une station à un point fixe ou d'une station aux points de détail. Les visées de type station-station ou station-point fixe (attribut REFERENCE=1) sont rouges avec une flèche indiquant la direction de la visée depuis la station, alors que les visées de point de détail (attribut REFERENCE=0) sont noires pointillées.

Symbologie restreinte

La symbologie restreinte ne garde que les visées de type station-station ou station-point fixe. Les points de détail ne sont pas représentés ici. Comme avant, on restreint au niveau de la couche directement, par le biais du *definition query* ("REFERENCE" =1).

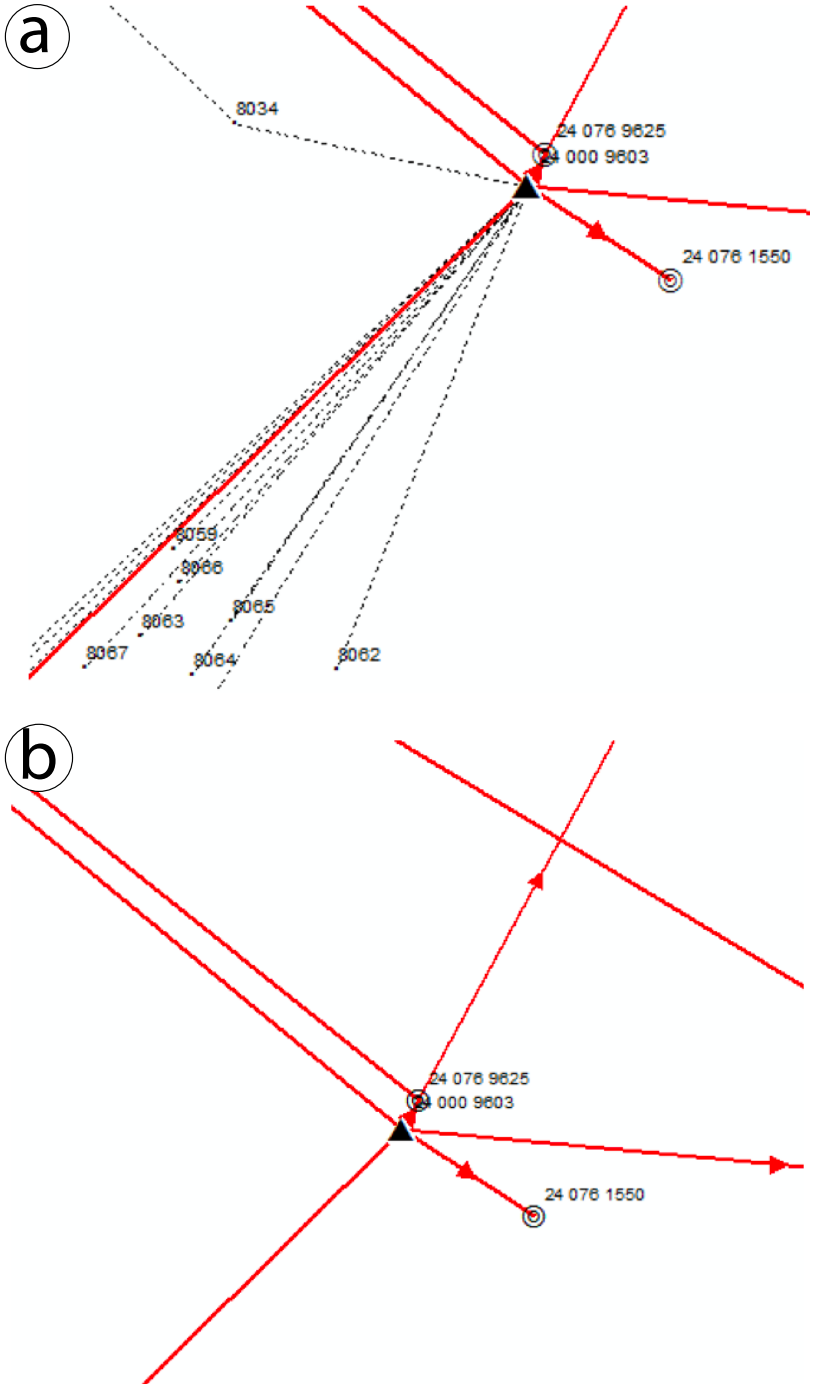


Figure 65 : Aperçu des deux types de symbologies pour les couches réseau_point et réseau_visée : **a)** générale et **b)** restreinte.

Finalement, il est possible d'intégrer ces fichiers *Layer Files* dans le fichier xml contenant les lyr des symbologies prédéfinies pour la barre d'outils de TopoGeo (voir 3.1). Il suffit de les inclure dans la symbologie souhaitée, et de charger ces couches (« réseau_point » et « réseau_visée ») dans un projet ArcMap.

4. Projets FME

FME (Feature Manipulation Engine) est un ETL (Extract-Transform-Load) spatial, un logiciel qui permet de transformer des données. Les données d'entrée (*reader*) et de sortie (*writer*) peuvent être sous différents formats (shapefile, tiff, ascii, las, gdb, mdb etc).

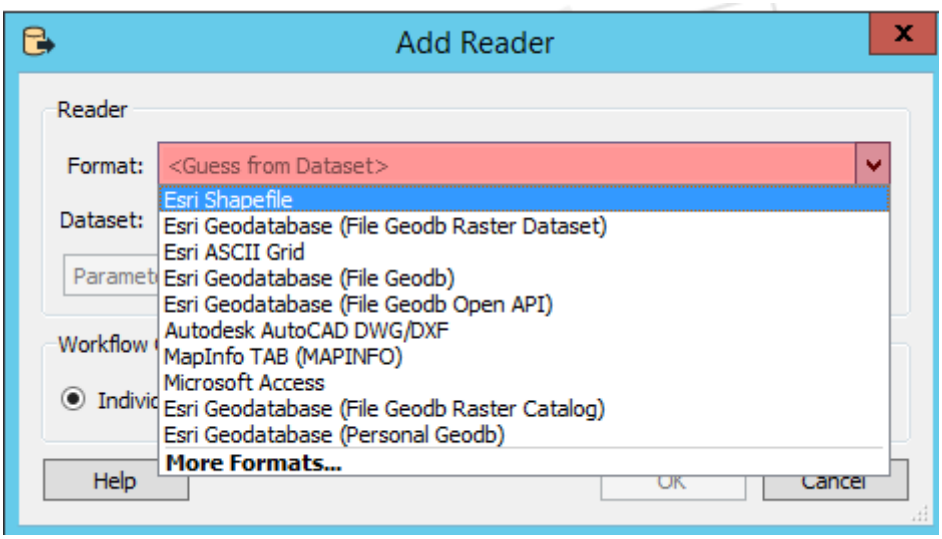


Figure 66 : Aperçu de quelques formats disponibles pour le reader.

Il est possible de créer des espaces de travail (*workspace*) dans l'interface du logiciel, qui est de type glisser-déposer. Cette méthode très intuitive sert à déplacer des éléments graphiques présents sur l'écran. Des *transformers* sont à disposition, qui sont les éléments de base permettant d'appliquer divers traitements à nos données, afin de les transformer. Chaque *transformer* a une fonction spécifique, et lorsqu'ils sont combinés, il est possible d'aboutir à des développements élaborés dans l'espace de travail (www.safe.com/transformers/).

Il est important de noter que la donnée en entrée reste toujours intacte, et il est possible de modifier les traitements qu'on lui applique à tout instant du développement. Cette fonctionnalité présente un grand avantage, que la plupart des logiciels SIG (ArcMap, ArcGIS Pro ou QGis) n'ont pas. En effet, ils suivent un cheminement de traitement irréversible ; si l'on applique plusieurs traitements à un shapefile, il est impossible de revenir en arrière et de modifier un paramètre que l'on avait oublié au départ. En revanche, sur FME, il est possible de visualiser les données à n'importe quel point de la transformation du flux de travail, sans enregistrer les fichiers. C'est le rôle des inspecteurs  , une fonctionnalité qui permet d'afficher les données internes du *transformer* ou du *reader*. Il suffit de connecter l'inspecteur à l'endroit où nous souhaitons obtenir des informations, et de charger le *workspace*. Cela nous donne une idée des résultats au fur et à mesure du développement (www.safe.com/fme/fme-desktop/).

FME est également un outil très pratique de conversion de données et il peut travailler sur de gros jeux de données. Lors d'une conversion, il suffit d'ajouter en tant que *reader* le fichier que l'on veut convertir, et en tant que *writer* le format dans lequel on souhaite le convertir.

Afin de rendre les développements plus lisibles, nous avons fait appel à des *bookmarks*, qui permettent de définir des zones dans l'espace de travail, afin de ranger les différentes étapes par catégories, pour faciliter l'accès au développement (Figure 67).

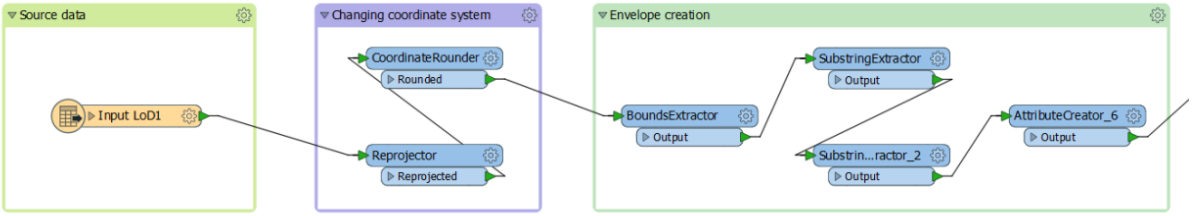


Figure 67 : Exemple de bookmarks, qui permettent d'organiser le développement (docs.safe.com).

Nous avons également utilisé des tunnels, qui permettent aux fils de liaisons d'être cachés, afin de ne pas surcharger l'interface du *workspace* (Figure 68).

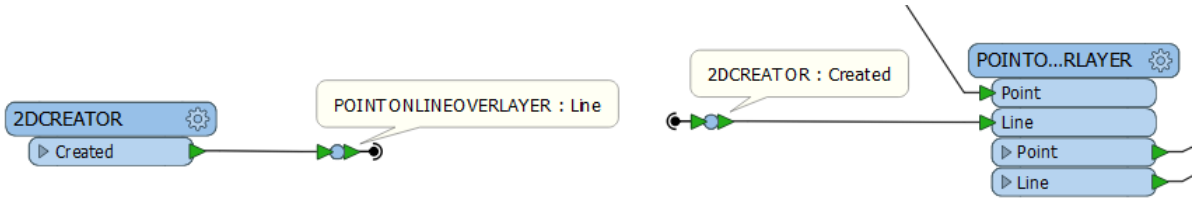


Figure 68 : Exemple de tunnel, qui permet de lier des transformers entre eux en cachant le lien pour rendre plus lisible le développement (docs.safe.com).

4.1 Projet division communes Carouge

4.1.1 But du projet

Le but de ce travail est de produire un shapefile et un fichier Excel par le biais de FME, en mettant en évidence les différentes parcelles touchées par un projet de division communale à Carouge. Il sera question de combiner chaque parcelle avec les quatre types de cession d'état ainsi qu'avec la couche de la couverture du sol OTEMO (provenant du SITG). Pour chaque parcelle, on veut calculer l'aire de chaque type de couverture du sol, et informer quels types de cession d'état sont intégrés dans ces surfaces. Cela permettra d'évaluer un coût potentiel par parcelle, en fonction du type de cession d'état et de la couverture du sol.

4.1.2 Données

Parcelles de la commune de Carouge : 2940, 2941, 2902, 2903, 2905 et 3207. Ces parcelles proviennent de la couche CAD_PARCELLE_MENSU (SITG).

Couverture du sol : surface verte, surface boisée, revêtement dur et bâtiments. La couche OTEMO (Ordonnance Technique du DDPS sur la Mensuration Officielle) est une ordonnance fédérale qui introduit un catalogue d'objets de couverture du sol cantonale, couvrant l'ensemble du canton de Genève. Elle distingue 6 genres de couverture du sol (CODE_OTEMO_1) et 25 niveaux sous-genres de couverture du sol (CODE_OTEMO_2) (https://ge.ch/sitg).

Types de cession d'état : dp communal, espace verdure, commune bâtiment G et commune autre périmètre. Cette couche provient du dwg fourni par le bureau.

NB : En ce qui concerne les types de cession d'état, le type « commune bâtiment G » est un polygone réservé à la construction, qui correspond à l'aire du droit à bâtir. Par conséquent, il ne devra pas être combiné à la couverture du sol (couche OTEMO).

4.1.3 Marche à suivre

1. AutoCAD : Il existe un projet contenant les parcelles impactées ainsi que les types de cession d'état (Figure 69a), dans le format natif des fichiers de dessins d'AutoCAD, le DWG (DraWinG). Nous souhaitons récupérer uniquement les contours des quatre types de cessions d'état du projet principal (encadrés en rouge dans la légende), il faut par conséquent les copier et les coller dans un nouveau projet. Ce nouveau fichier dwg contient 13 entités au total (Figure 69b).

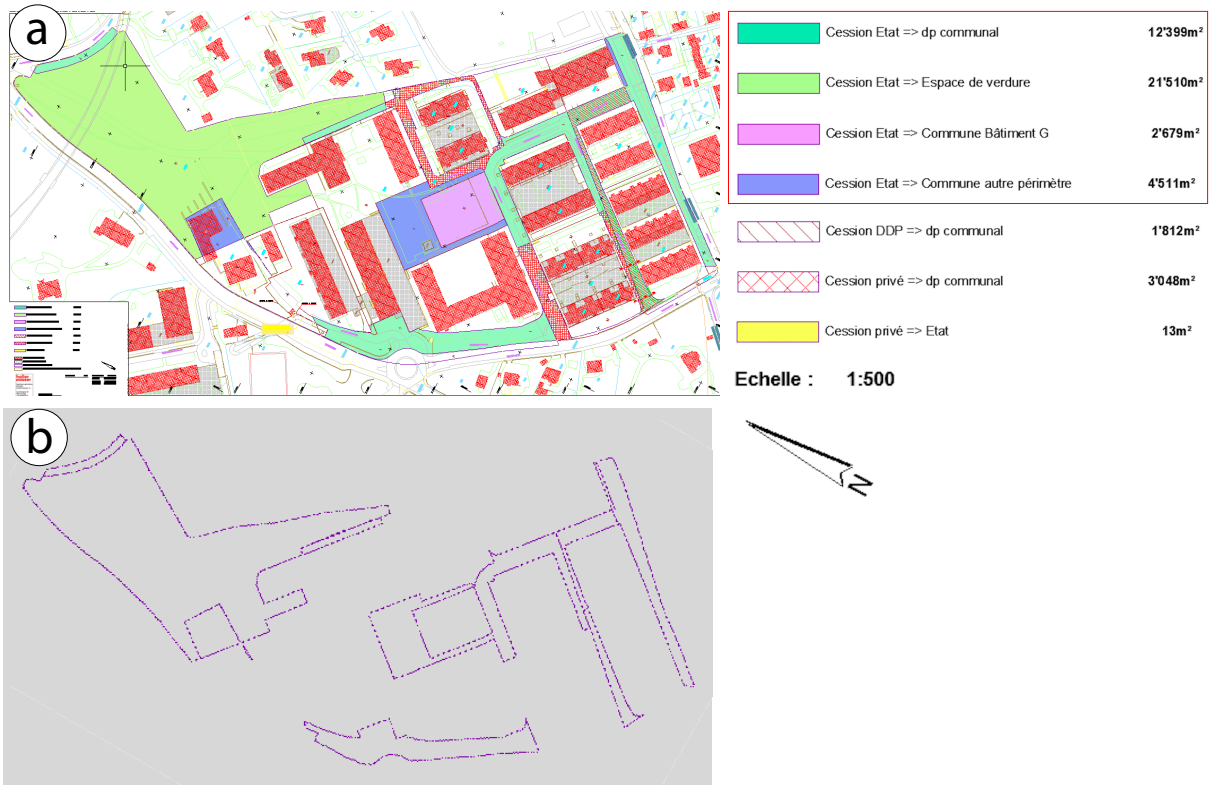


Figure 69 : Projet sur AutoCAD avec **a)** tous les types de cession d'état représentés, avec leur aire et **b)** une copie des polygones des quatre premiers types de cession d'état dans un nouveau projet.

2. FME : Dans un nouveau projet FME, il faut convertir ce fichier dwg en shapefile (« shp_cession_etat »), pour pouvoir travailler sur sa table attributaire par la suite. Pour ce faire, nous ajoutons un *reader* (le dwg) et un *writer* (le shp), nous les lions entre eux et lançons le *workspace*.

3. QGis : Nous chargeons le shapefile fraîchement créé dans un nouveau projet QGis et éditons la couche pour pouvoir ajouter un nouveau champ « type » dans la table attributaire. Ce champ va nous permettre d'indiquer quel est le type de cession d'état pour chaque polygone, car cet attribut n'est pas maintenu depuis le dwg.

4. FME : Dans un nouveau projet, il faut charger cette couche shapefile (« shp_cession_etat ») ainsi que le shapefile des parcelles et celui de la carte OTEMO en tant que *readers*.

Transformers

Tester
Test Filter
Clipper
Area Calculator
Aggregator

Dans un premier temps, nous souhaitons isoler les parcelles qui nous intéressent. Il suffit d'appliquer un ***Tester*** à la couche des parcelles « CAD_PARCELLE_MENSU », pour isoler la commune de Carouge. Ce *transformer* renvoie deux résultats : *Passed* et *Failed*. Le *passed* correspond à la requête que nous avons faite, et le *failed* correspond à toutes les autres entités. Puis, nous ajoutons un ***TestFilter*** pour isoler les parcelles 2940, 2941, 2902, 2903, 2905 et 3207 (Figure 70). Ce *transformer* nous permet de filtrer les résultats en fonction de ce que nous cherchons à mettre en évidence.

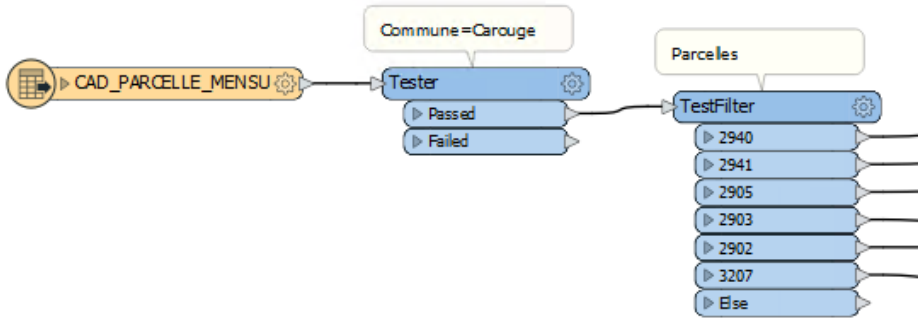


Figure 70 : Transformers appliqués à la couche des parcelles (SITG), afin d'isoler les parcelles désirées de la commune de Carouge.

Comme nous ne souhaitons pas que la zone « commune bâtiment G » soit combinée à la couche OTEMO, nous allons la traiter séparément. Il faut donc appliquer un ***TestFilter*** au shapefile « shp_cession_etat », pour séparer « batiment_G » des autres types de cession d'état. Ensuite nous lui appliquons un ***AreaCalculator***, qui va créer un nouvel attribut « _area », afin de connaître la géométrie de ce polygone. Finalement, la sortie de cette couche sera liée à un shapefile qui lui est propre (« droit_batir »), mais également au shapefile et au fichier Excel qui contiennent les informations des autres zones de cession. Ceci a pour but de comptabiliser l'aire du polygone « batiment_G » dans la somme finale. Le reste (*Unfiltered*), correspond aux trois autres types de cession d'état, *i.e.* dp communal, espace verdure et commune autre périmètre (Figure 71).

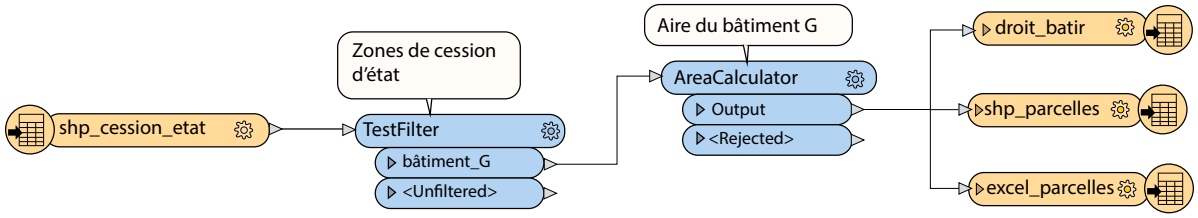


Figure 71 : Transformers appliqués à la couche shp_cession_état, préalablement créée sur FME et modifiée sur QGis, afin de séparer les zones de cession d'état en deux groupes ; batiment_G et le reste (dp_communal, espace_verdure et commune_autre_perimetre). Les données en sortie de la branche batiment_G sont liées à trois writers : droit_batir, shp_parcelles et excel_parcelles.

Ce *Unfiltered* va être découpé en fonction de chaque parcelle, de manière individuelle, par le biais du *transformer* ***Clipper***. Ce *transformer* effectue une opération de découpage géométrique, où la plupart des types de géométrie peuvent être découpés par une zone, et les attributs peuvent être partagés entre les objets (joint spatial). Il contient deux branches : le *clipper* et le *clippee*. Le *clipper* est la géométrie principale, qui fait office de masque pour délimiter la zone à étudier. Le *clippee* correspond aux géométries inclues dans la géométrie principale. Ici, le *clipper* est la parcelle et le *clippee* la zone de cession d'état.

Ensuite, un nouveau ***Clipper*** est appliqué afin de découper la couche « CAD_CARTE_OTEMO » en fonction de la sortie de la couche précédente (parcelle vs. cession d'état). Le *clipper* est la sortie précédente, et le *clippee* est la couche de la couverture du sol (Figure 72).

La couche en sortie de ces deux *clippers* sera un découpage par parcelle, contenant les contours des zones de cessions d'état, qui eux-mêmes seront classifiés en fonction de la légende OTEMO. En revanche, cette classification contient plusieurs petits polygones, qu'il faudra regrouper par la suite.

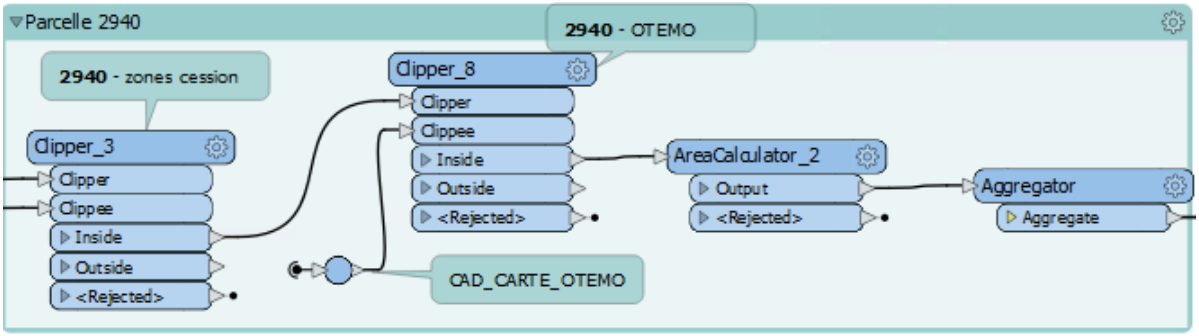


Figure 72 : Exemple de transformers appliqués à la parcelle 2940. **Clipper_3** : opère un découpage des zones de cession d'état en fonction de la parcelle**. Clipper_8** : opère un découpage de la couche OTEMO par la parcelle. **AreaCalculator2** : calcule l'aire de tous les polygones sortis. **Aggregator** : regroupe les polygones par le type de couverture du sol et additionne la somme des aires des polygones.

Nous appliquons un ***AreaCalculator*** afin d'obtenir l'aire de chacun de ces polygones, puis un ***Aggregator***, afin d'additionner les aires de ces polygones. Ce *transformer* va combiner des géométries en les agrégeant, et permet d'effectuer quelques opérations (concaténation, somme, moyenne). Dans notre cas de figure, il est nécessaire de grouper les informations par attributs (*Group by*), en l'occurrence par la couverture du sol (« LEG_OTEMO »), autrement tous les attributs seront considérés. Puis nous précisons que nous voulons concaténer l'attribut « type », qui est l'attribut créé par nos soins sur QGis définissant les zones de cession d'état. Finalement, nous additionnons

l'aire de tous les petits polygones, afin d'obtenir une aire par couverture du sol, en indiquant quels types de cession d'état sont inclus dans ces aires.

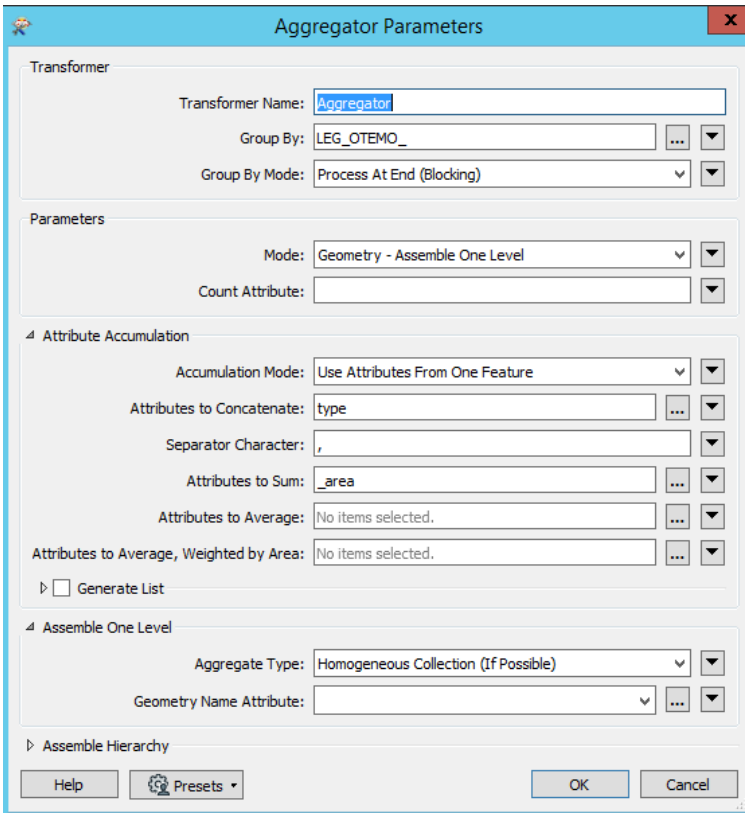


Figure 73 : Paramètres du transformer « Aggregator ». Groupement par l'attribut LEG_OTEMO_, concaténation de l'attribut type et somme de l'attribut _area.

Finalement, il faut créer deux *writers*, un pour le fichier Excel et un pour le shapefile. Il faudra rajouter les attributs « _area » et « type » dans les *Users attributes* du *writer*, car ils ne se mettront pas automatiquement (Figure 74).

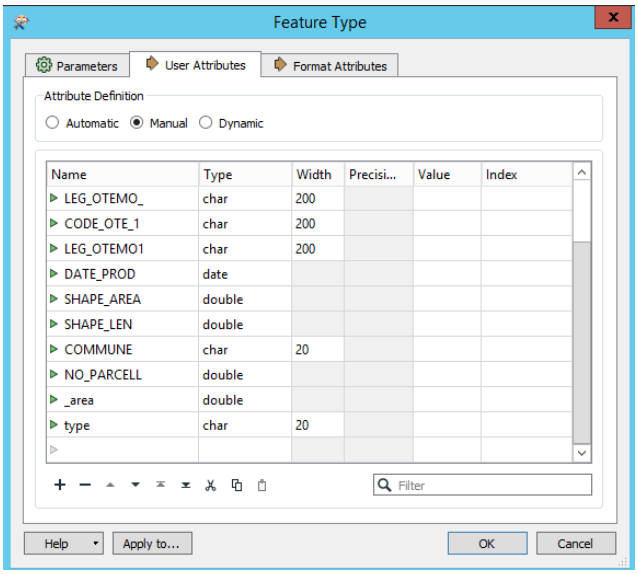


Figure 74 : Attributs à rajouter dans la couche shapefile et le fichier Excel en sortie.

5. ArcMap : Il faut ajouter la couche fraîchement créée, et changer sa symbologie en fonction de la couverture du sol (« LEG_OTEMO »). Il est préférable d'ajouter la couche des parcelles pour visualiser aussi les parcelles autour de la zone d'intérêt.

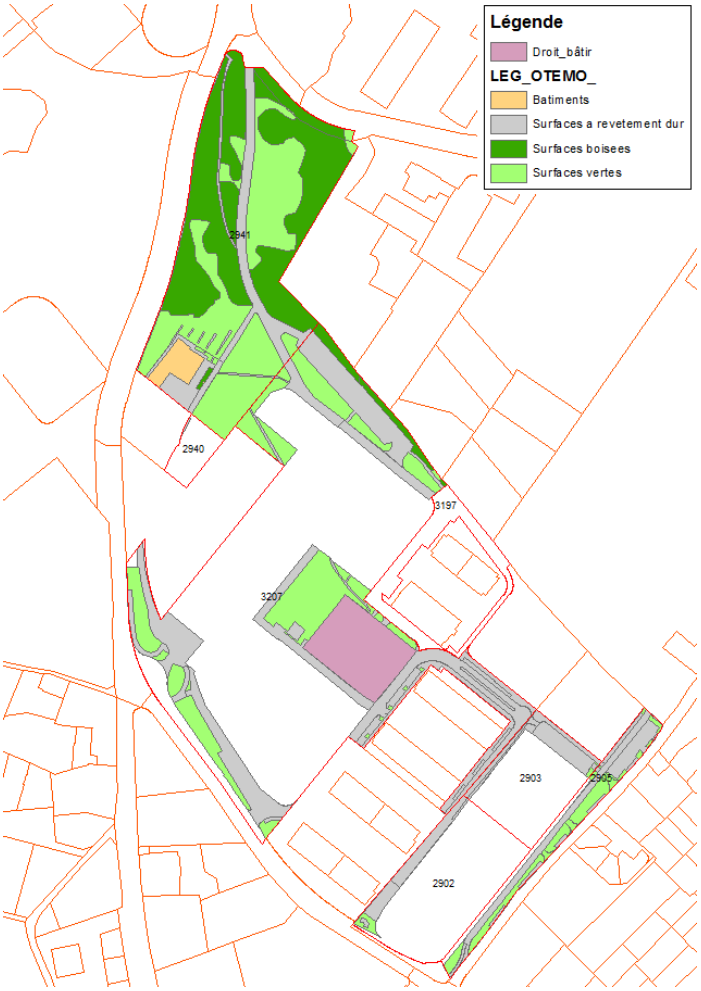


Figure 75 : Carte en sortie sur ArcMap. Les numéros correspondent aux numéros de parcelles impactées. Les polygones avec une symbologie sont les contours des trois types de cession d'état (dp communal, espace_verdure et commune_autre_perimetre) qui contiennent les différentes couvertures du sol. Le polygone rose « droit_bâtir » correspond au polygone du bâtiment G, qui est l'aire constructible, pas symbolisée avec la couverture du sol.

LEG_OTEMO_	COMMUNE	NO_PARCELL	_area	type
Batiments	Carouge	2941	596	autre_peri
Surfaces vertes	Carouge	2941	7227	verdure,autre_peri,dp_commu
Surfaces boisees	Carouge	2941	8706	verdure,autre_peri,dp_commu
Surfaces a revetement dur	Carouge	2941	2611	verdure,autre_peri,dp_commu
Surfaces vertes	Carouge	2905	989	autre_peri,dp_commu
Surfaces a revetement dur	Carouge	2905	1555	autre_peri,dp_commu
Surfaces vertes	Carouge	2902	94	dp_commu
Surfaces a revetement dur	Carouge	2902	734	dp_commu
Surfaces vertes	Carouge	2903	23	dp_commu
Surfaces a revetement dur	Carouge	2903	962	dp_commu
Surfaces a revetement dur	Carouge	3207	8494	verdure,autre_peri,dp_commu
Surfaces vertes	Carouge	3207	5290	verdure,autre_peri,dp_commu
Surfaces boisees	Carouge	3207	494	verdure
	Carouge	3207	2679	batiment_G
			somme	
			40454	

Figure 76 : Fichier Excel obtenu, avec les différentes couvertures du sol, types de cession d'état et aires calculées par parcelle.

4.2 Grille de points contenant les altitudes des modèles numériques

4.2.1 But du projet

Ce projet a deux finalités différentes, avec des processus très similaires. Nous souhaitons créer une grille de points sur les modèles numériques au format gdb et datant de 2019, en indiquant la hauteur de chaque point (pixel) de la grille. Nous ne possédons que le MNT et le MNA, le MNS est manquant.

Dans le premier cas, nous souhaitons créer une grille avec des points séparés entre eux de 100 mètres. Cette grille, au format shapefile, sera chargée dans le projet QGis et affichée par-dessus les modèles numériques, afin d'obtenir visuellement des valeurs de hauteur. Cependant, ces points sont très éloignés entre eux, ce qui n'est pas utile lorsque l'on souhaite faire un contrôle sur un bâtiment ou sur un petit objet.

De ce fait, dans le deuxième cas, nous créons une grille de points séparés de 2 mètre entre eux. L'inconvénient de cette grille est qu'elle est beaucoup plus volumineuse et dense, ce qui ralentit passablement le projet QGis. C'est la raison pour laquelle nous allons créer une action sur QGis permettant de charger les shapefiles contenant cette grille de points séparés de 2 mètres. La grille sera découpée en fonction du shapefile des tuiles des orthophotos qui englobent le canton de Genève (voir point 2.1.2).

4.2.2 Grille de points (100 mètres)

Lors de nos traitements sur FME, nous avons utilisé des petites extractions de modèles numériques, téléchargées sur le SITG, et non la gdb entière. Ceci a pour but de pouvoir faire plusieurs tests sur les données, sans que FME ne mouline trop. Une fois le développement approuvé, nous l'appliquons sur la donnée entière.

Transformers

2D Grid Accumulator
Point On Raster Value Extractor
List Indexer
Null Attribute Mapper
Tester

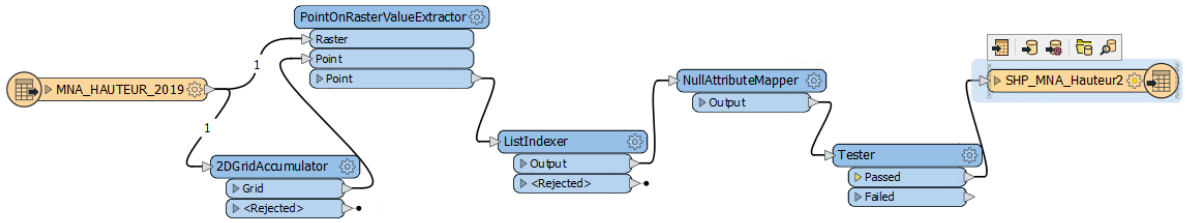


Figure 77 : Développement sur FME avec les différents transformers utilisés pour obtenir la grille de points séparés de 100 mètres, au format shapefile.

Sur FME, on charge notre *reader*, qui est le MNT ou le MNA. On applique une grille ***2DGridAccumulator*** directement au raster (le *reader*), qui aura la même emprise que celui-ci. Ce *transformer* remplace les éléments d'entrée par une grille d'éléments ponctuels ou polygonaux

bidimensionnels ayant l'espacement spécifié. Concernant le type de grille, nous avons choisi de placer les points au centre de chaque carré (*centers*), et non aux extrémités de ceux-ci (*corners*), car cela est problématique lorsque l'on souhaite comparer la donnée du pixel du raster et la donnée du point créé. En effet, la donnée du point sera basée sur l'intersection de 4 pixels, et donc la valeur affichée sera celle d'un des 4 points, alors que nous souhaitons avoir une valeur propre à un pixel. En revanche, le fait de placer les points au centre ne pose pas de problème de superposition de pixels, mais possède aussi un inconvénient, moindre et acceptable : celui de ne pas prendre en compte les coordonnées aux extrémités du cadre.

Dans la catégorie *Cell Size* (Figure 78), on détermine la longueur de la colonne et la hauteur de la ligne. Dans notre cas, il s'agit d'une grille de 100x100.

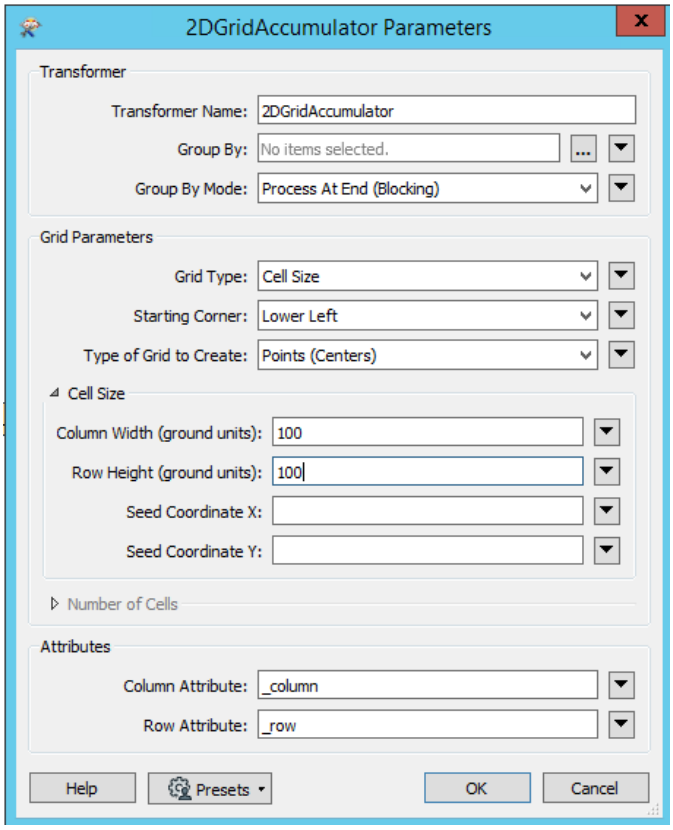


Figure 78 : Paramètres du transformer 2DGridAccumulator. Le type de grille est en fonction de la taille de la cellule (cell size), avec les points au centre de la cellule, et la taille de la cellule est de 100x100.

Ensuite on applique un ***PointOnRasterValueExtractor***, qui va extraire les valeurs de bande et de palette du raster (ici le modèle numérique) à l'emplacement d'un ou plusieurs points d'entrée (ici la grille) et les définit comme attributs sur l'élément. Ce *transformer*, qui contient deux entrées (raster et point), est généralement utilisé pour extraire les élévations d'un modèle numérique et leur attribuer des points. Ainsi, une couche issue de la superposition du raster initial et de la grille va se créer. Les valeurs sont interpolées et ajoutées à un nouvel attribut de liste, nommé _bande{}.valeur et _bande{}.palette{}.valeur, respectivement (http://docs.safe.com/).

Les attributs de cette superposition vont apparaître sous forme de liste, ce qui n'est pas exactement ce que l'on souhaite obtenir. On cherche à récupérer, pour chaque point de la grille, l'altitude z du point, grâce aux altitudes du modèle numérique. Le souci est que le raster du modèle numérique n'a

qu'un attribut, la band{0}, car c'est une couche de type mono-bande. Ces couches représentent soit des variables continues (*par exemple* : élévation) soit des variables discrètes (*par exemple* : utilisation du sol) (https://docs.qgis.org/2.14/fr/docs/pyqgis_developer_cookbook/raster.html).

C'est pourquoi il faut appliquer un *transformer* supplémentaire, comme par exemple un ***ListExploder*** ou un ***ListIndexer***. Le ***ListExploder*** va scinder la liste par nom d'attribut et par index, en rajoutant une colonne « value » et « index name ». Dans notre cas de figure, on a qu'un seul attribut, la band{0}, qui correspond à la hauteur z. Le ***ListIndexer*** donne le nom de l'index et la valeur à lui attribuer (ici 0 car il s'agit de la band{0}). Ce *transformer* copie les attributs de l'élément spécifié par l'index pour devenir les attributs principaux de l'élément (http://docs.safe.com/). Ici aussi un nouveau champ « value » est créé dans la table attributaire, contenant les valeurs de l'altitude z.

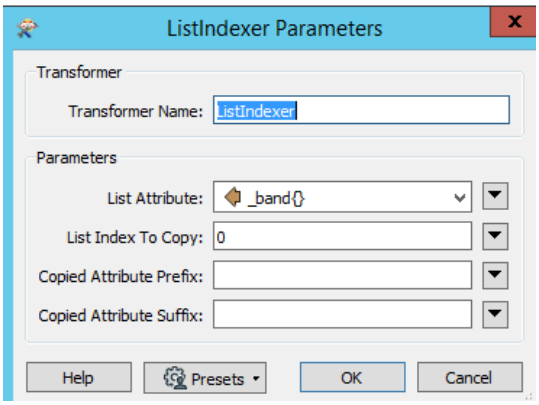


Figure 79 : Paramètres du transformer ListIndexer.

Il se peut que certains points soient en dehors de notre cadre, et ils auront par conséquent une valeur nulle. Afin de faire disparaître ces points, il faut appliquer un ***NullAttributeMapper*** (Figure 80), qui va mettre en évidence ces valeurs. Nous décidons d'appliquer le terme *Missing* aux valeurs nulles.

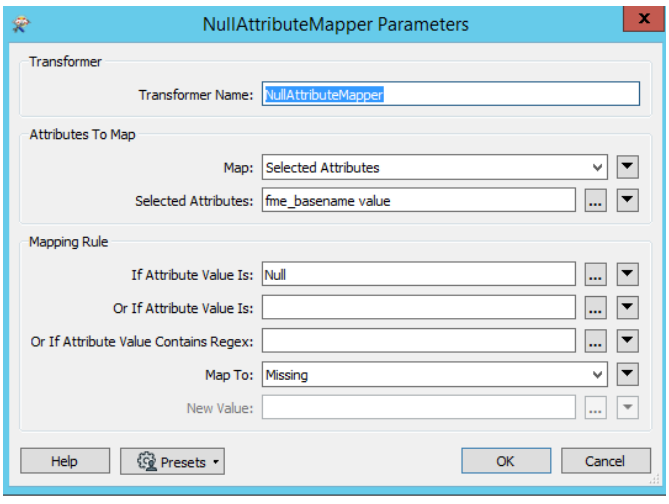


Figure 80 : Paramètres du transformer NullAttributeMapper. Lorsque la valeur d'un attribut est nulle, on applique le terme Missing, qui nous permettra de filtrer les valeurs par la suite.

Puis nous appliquons un ***Tester*** (Figure 81). Comme les attributs « fme_basename » et « value » peuvent contenir des valeurs nulles, nous précisons que nous ne voulons pas les entités de ces

attributs qui ont le terme *Missing* (*Operator : attribute is missing*). Le ***Tester*** va renvoyer deux sorties ; une sans valeurs nulles (*passed*) qui nous intéresse, et une avec les valeurs nulles (*failed*) (Figure 77). Nous aurions aussi pu créer une expression sans la négation et garder les valeurs *failed*, qui auraient été les valeurs non nulles.

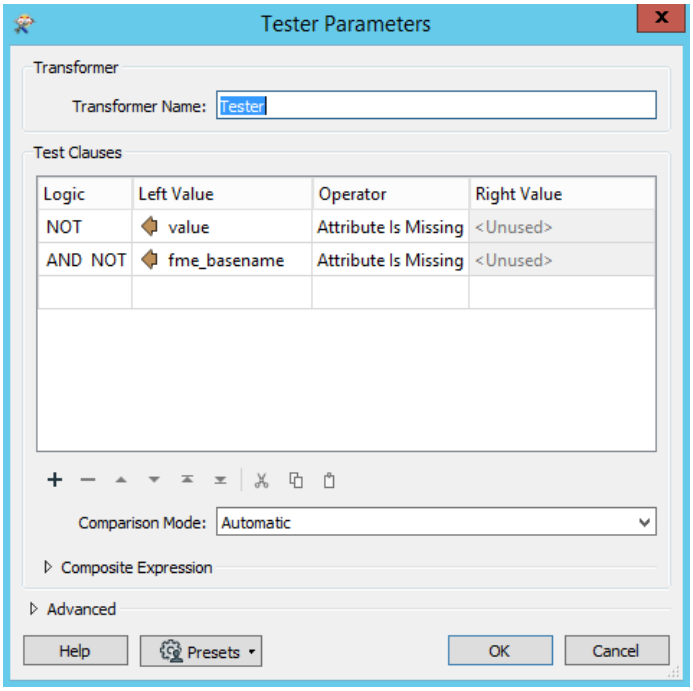


Figure 81 : Paramètres du transformer Tester, visant à éliminer les valeurs nulles dans les attributs « value » et « fme_basename ».

Le *writer* est sous forme de shapefile, et il faut s'assurer de bien rentrer l'attribut « value » (altitude z) dans la table attributaire du shapefile sortant, ainsi que de préciser shapefile_point dans la géométrie.

NB : Ici on a utilisé le transformer ***PointOnRasterValueExtractor***, qui superpose un raster et un shapefile sous forme de points. Ce *transformer* est similaire au ***Overlayer***, qui superpose deux couches vecteur, de type point, ligne, surface et aire.

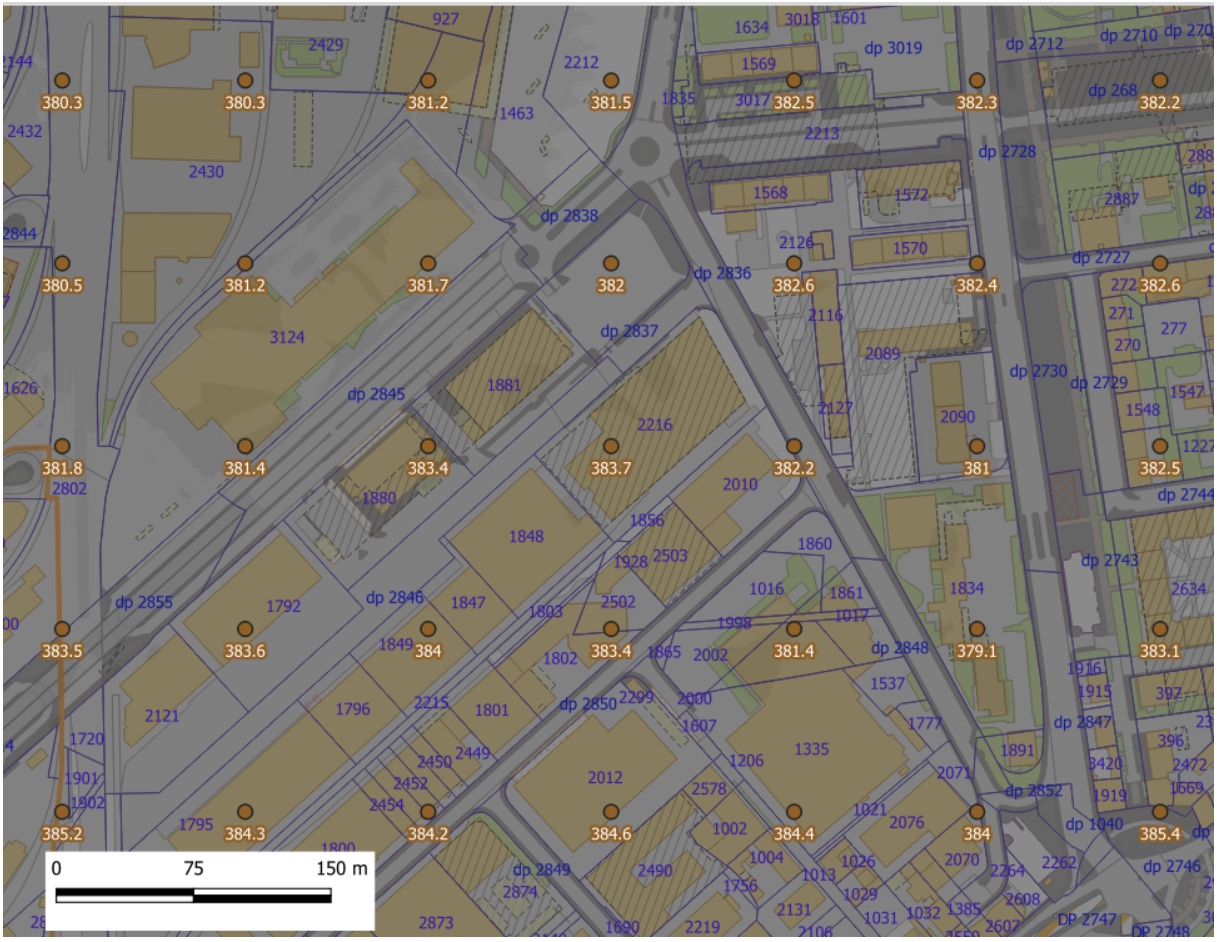


Figure 82 : Aperçu du résultat de la grille shapefile de points séparés de 100 mètres, sur QGis.

Autre méthode

Il est possible de redéfinir la taille des cellules (pixel) du raster entrant (ici le modèle numérique), par le biais de l'outil ***RasterResampler***, en définissant un nouvel espacement de cellules. Puis, il faut convertir ce raster en polygones par le biais d'un ***RasterToPolygonCoercer***. Finalement, pour obtenir un point centré et non un polygone, il faut utiliser le *transformer* ***CenterPointReplacer***. Cependant, nous avons remarqué que cette méthode est beaucoup moins précise ; la hauteur du point et celle du raster du modèle numérique ne sont pas complètement identiques.

4.2.3 Grille de points (2 mètres)

Le processus est très similaire au précédent, avec quelques *transformers* en plus.

Les données à disposition sont :

- Le MNT et le MNA sous forme de gdb.

- Un shapefile du canton de Genève, découpé en 436 polygones.

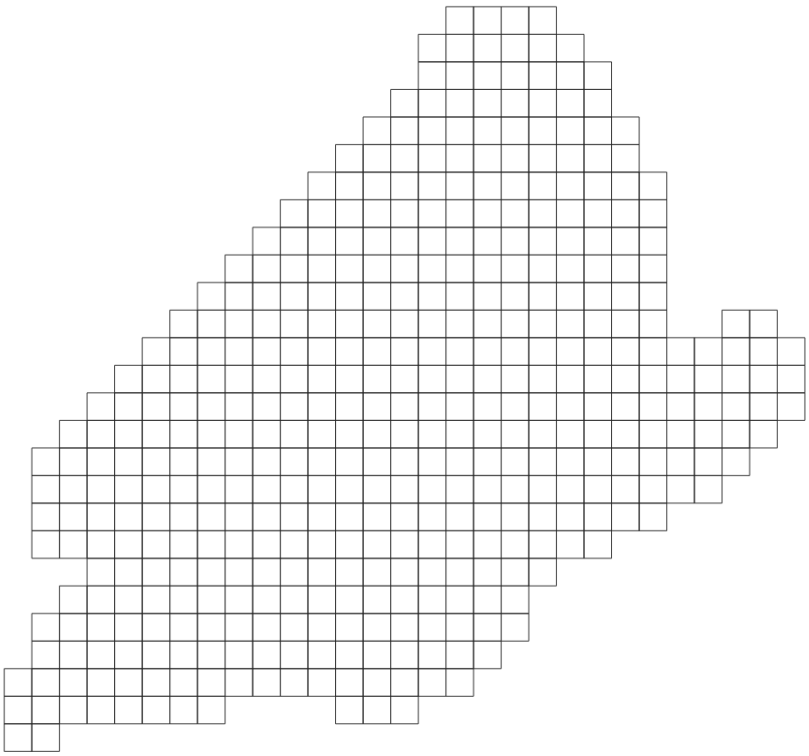


Figure 83 : Shapefile du canton de Genève découpé en polygones (tuiles orthophoto).

4.2.4 Traitement des données sur FME

Transformers

2D Grid Accumulator
Point On Raster Value Extractor
List Indexer
Null Attribute Mapper
Tester
String Replacer
Test Filter
Point On Area Overlayer

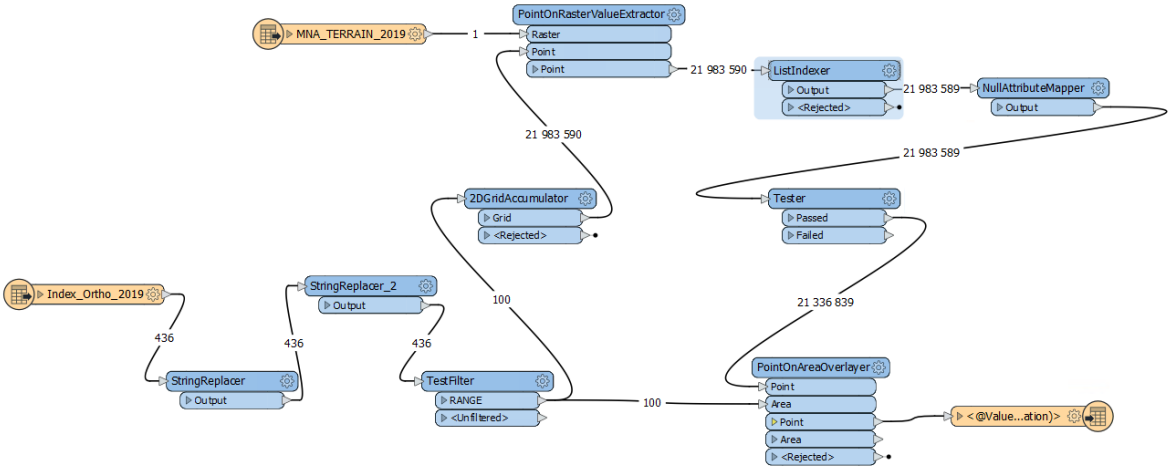


Figure 84 : Développement FME avec les divers readers, transformers et writer, afin de créer une grille de points séparés de 2 mètres, au format shapefile.

Il faut ajouter deux *readers* ; l'index des orthophotos créé sur QGis (shapefile : « Index_ortho_2019 ») et le MNT (geodatabase : « MNA_TERRAIN_2019 ») (ou le MNA).

Le shapefile des orthophotos va nous être utile sur FME, car il fera office de masque pour découper le MNT/MNA en polygones. Dans la table attributaire de ce shapefile, il y a le chemin de chacune des tuiles sous le nom du champ attribué sur QGis (« location »).

Commençons par l'index orthophoto ; l'objectif est de modifier son attribut « location », qui est l'attribut du chemin entier des différentes tuiles, afin de ne garder que le numéro unique de chaque tif qui est la concaténation de ses coordonnées X et Y (par exemple : 24851109). Ceci permet d'avoir un shapefile qui n'est pas lié à un chemin en particulier, et qui pourra par conséquent être déplacé dans l'ordinateur.

Pour ce faire, nous utilisons le *transformer* ***StringReplacer***, qui permet de modifier les attributs du shapefile, en remplaçant tout le chemin (à l'exception du numéro du tif) par du vide. Ensuite nous appliquons un ***TestFilter***, dans le but de ne sélectionner qu'une centaine de tuiles, pour ne pas surcharger le développement (Figure 85).

D'autre part, pour éviter que le développement ne prenne trop de place sur le disque C, il est possible de désactiver le *Feature caching* dans l'onglet *Run*. Lorsqu'il est activé, ce paramètre permet de stocker les données afin d'y accéder facilement en cas de besoin. Une loupe apparait à côté des *transformers*, et FME met en cache les données lors de leur traitement. Cela est utile lorsque l'on souhaite contrôler les résultats de notre développement à tout moment. En revanche, lorsque l'on est satisfait du développement et que l'on souhaite charger l'espace de travail, il n'est pas nécessaire d'activer les caches (https://www.safe.com/blog/2018/05/caching-data-fme-evangelist174/).

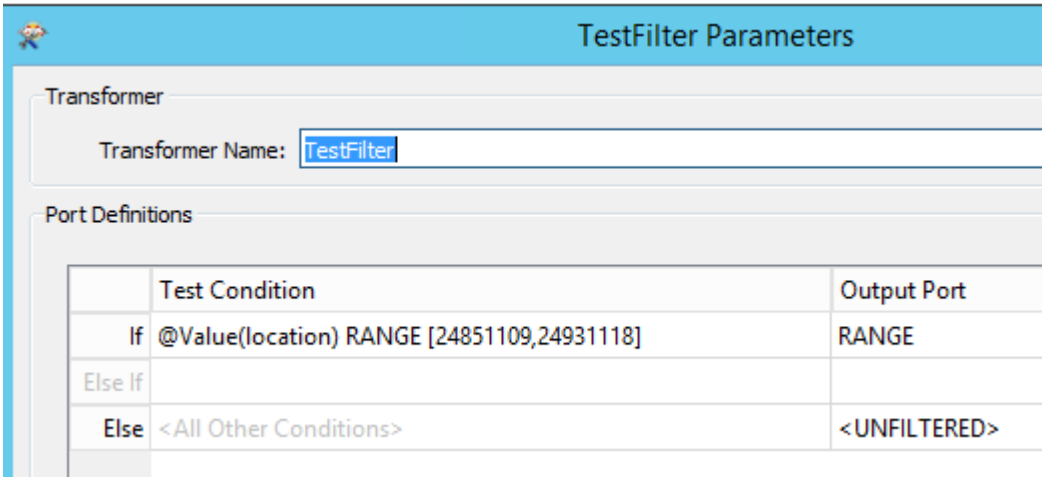


Figure 85 : Paramètres du transformer TestFilter, en triant les tuiles par gamme (RANGE).

En parallèle, nous appliquons des *transformers* au MNT. L'objectif est de lui appliquer une grille de points, séparés de 2 mètres, en attribuant l'altitude du MNT à ces points. L'outil permettant de créer la grille est le ***2DGridAccumulator***.

NB : Vu que nous avons lié la grille au shapefile et au MNT, elle ne va pas interroger l'intégralité du MNT, mais juste la partie qui est recoupée par le shapefile du canton.

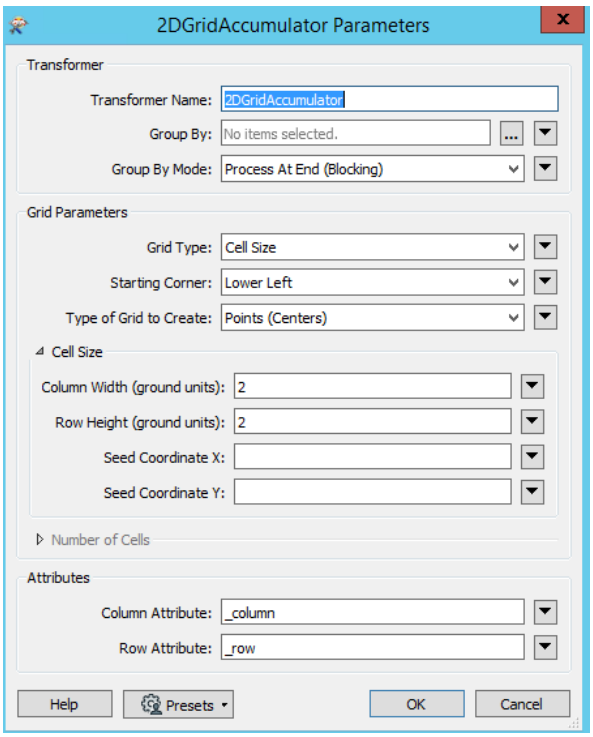


Figure 86 : Paramètres du transformer 2DGridAccumulator. Le type de grille est en fonction de la taille de la cellule (cell size), avec les points au centre de la cellule, et la taille de la cellule est de 2x2.

Puis, comme pour le point précédent, nous appliquons un ***PointOnRasterValueExtractor***, puis un ***ListIndexer***, ***NullAttributeMapper*** et ***Tester.***

Ensuite, nous créons notre masque sur le MNT par rapport au shapefile du canton de Genève découpé en polygones, par le biais du *transformer* ***PointOnAreaOverlayer*** (Figure 87). Ce *transformer*

permet d'effectuer une jointure spatiale, en superposant des points et des polygones ; chaque point reçoit les attributs de la zone dans laquelle il est contenu, et chaque zone possédant un ou plusieurs points reçoit les attributs de chaque point (https://www.safe.com/transformers/).

Il faut s'assurer que le paramètre *Merge Attributes* soit coché, sinon, la combinaison d'attributs n'a pas lieu. La sortie qui nous intéresse est celle des points, que l'on lie directement au *writer* (Figure 84).

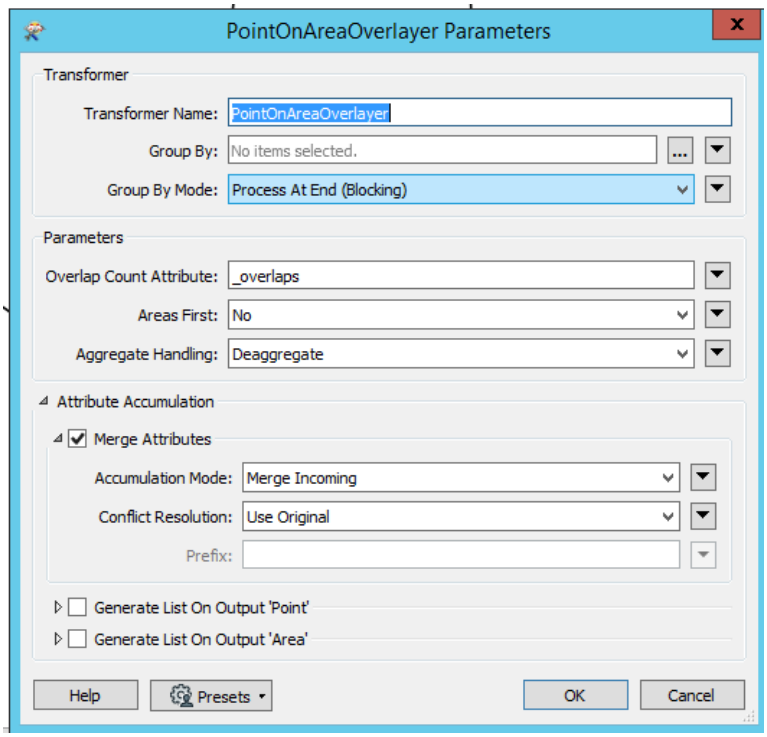


Figure 87 : Paramètres du transformer PointOnAreaOverlayer, permettant de superposer les points de la grille dans les polygones de la couche shapefile du contour du canton de Genève.

Le *writer* sera un shapefile unique, contenant toutes les tuiles. En double-cliquant dessus, il est possible de lui rajouter des attributs qui n'ont pas été considérés automatiquement, et il est également possible de définir quel sera le nom du shapefile. Nous lui attribuons le nom de l'attribut « location » (Figure 88), qui correspond au numéro du tif.

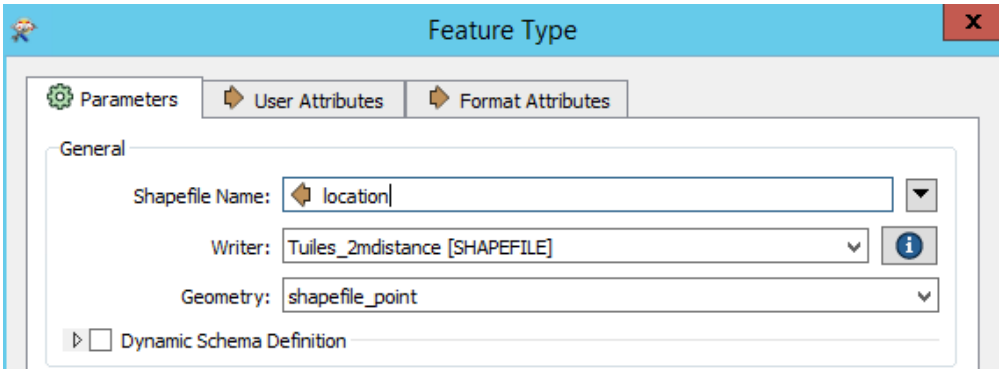


Figure 88 : Paramètres du shapefile en sortie. Le nom du shapefile est en fonction de l'attribut « location », qui est le chemin des tuiles. On précise que l'on souhaite obtenir une géométrie de type point pour le shapefile en sortie.

Afin de générer plusieurs shapefiles distincts en fonction de chaque tuile, nous devons cocher le *Fanout Dataset* et lui indiquer le chemin où nous voulons enregistrer nos shapefiles (Figure 89). Le *Fanout Dataset* permet de diviser les données de sortie en fonction de la valeur d'un attribut (ici « location ») (http://docs.safe.com).

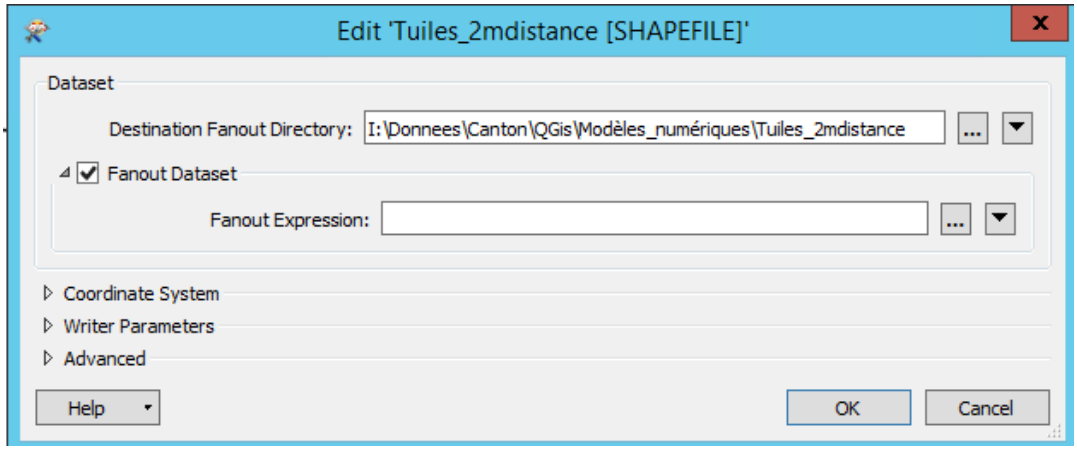


Figure 89 : Fonction « fanout dataset », permettant de répartir le shapefile en plusieurs shapefiles distincts dans le chemin indiqué.

L'inconvénient est que le shapefile généré n'a pas de symbologie particulière ni d'étiquettes. Or, nous souhaitons attribuer des étiquettes aux points de la grille, en affichant leur altitude, pour qu'elles apparaissent sur QGis. Nous avons essayé le *transformer* ***LabelPointReplacer***, qui remplace la géométrie de l'entité par un point d'étiquette, en affichant la valeur du champ qu'on lui attribue. Cependant, lorsque l'on charge le shapefile sur QGis, les étiquettes ne sont pas maintenues. Par conséquent, nous devrons coder dans l'action sur QGis pour ouvrir un style préenregistré.

4.2.5 Action sur QGis

Il faut tout d'abord charger le shapefile de l'index des tuiles (« Index_ortho_2019 ») dans le projet. C'est à cette couche que nous allons appliquer l'action pour charger les différentes tuiles. L'action est similaire à celle des dalles orthophotos, sauf qu'elle ouvre un vecteur et non un raster.

De plus, pour charger un style, nous devons d'abord l'enregistrer sous format .qml. Puis, dans le code, il faut indiquer le chemin du style et à quel groupe de couche on souhaite l'appliquer, en l'occurrence, les shapefiles contenant la grille de points séparés de 2 mètres.

Code

```
import os  # On charge le paquet os.
import ntpath  # On importe le module pathname (WindowsNT/95 version).
def getVectorLayerByName(NomCouche):
    layermap=QgsProject.instance().mapLayers()()# Il faut que la couche se trouve dans l'arborescence du projet ouvert.
    for name, layer in layermap.items():
        if layer.name()==NomCouche:
            if layer.isValid(): # On teste la validité de la couche.
               return layer
            else:
               return None
```

```
mypath=r"I:\Donnees\Canton\QGis\Modèles_numériques\Tuiles_2mdistance\[%location%].shp"  # [%location%] correspond à l'attribut où sont stockés les numéros des coordonnées des tuiles. Nous rajoutons le chemin, pour que le logiciel sache où trouver les shp.
instRegistry = QgsProject.instance()
nom=ntpath.basename(mypath)  # Crée le lien avec le chemin indiqué auparavant (basename : renvoie la composante finale d'un nom de chemin).

vector_ouvert=getVectorLayerByName(nom)  #On charge le vecteur (en l'occurrence le shp de la tuile) lors du clic.

if vector_ouvert is not None:
    QgsProject.instance().removeMapLayer(vector_ouvert.id())
    qgis.utils.iface.mapCanvas().refresh()
else:
    vector_ouvert = QgsVectorLayer(mypath,nom)

    QgsProject.instance().addMapLayer(vector_ouvert, False)
    root = QgsProject.instance().layerTreeRoot()#On charge le shp dans arborescence des couches.
    g = root.findGroup("MNT_grille_2m")  #Et on précise qu'on veut le charger sous le groupe « MNT_grille_2m».
    g.insertChildNode(0, QgsLayerTreeLayer(vector_ouvert))  #adds a node (single list element) to the node's array of children in the scene graph hierarchy.

from qgis.core import *  #On importe tout dans le module qgis.core.
def applystyle_group(name):
    root = QgsProject.instance().layerTreeRoot()#Cela s'applique au projet actuel, dans l'arborescence.
    point = root.findGroup(name)  #Trouver le groupe qu'on définit plus bas (MNT_grille_2m).
    for child in point.children():
        if isinstance(child, QgsLayerTreeLayer):
            if child.layer().type()==0:

child.layer().loadNamedStyle(r"I:\Donnees\Canton\QGis\Styles_QGis\grille_2m_MNT.qml")#On indique le chemin du style à appliquer.
applystyle_group("MNT_grille_2m")#On indique à quel groupe de couches on souhaite appliquer le style.
```

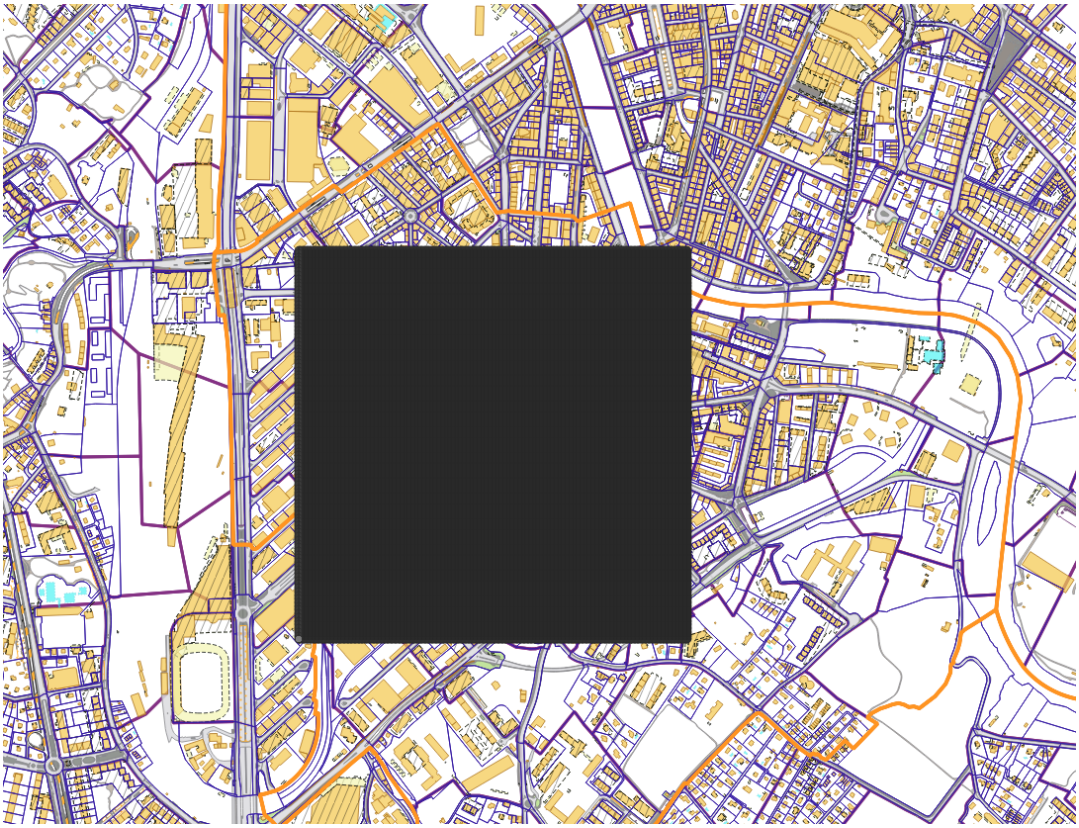


Figure 90 : Grille de points séparés de 2 mètres de la zone (tuile) cliquée, sur QGis.

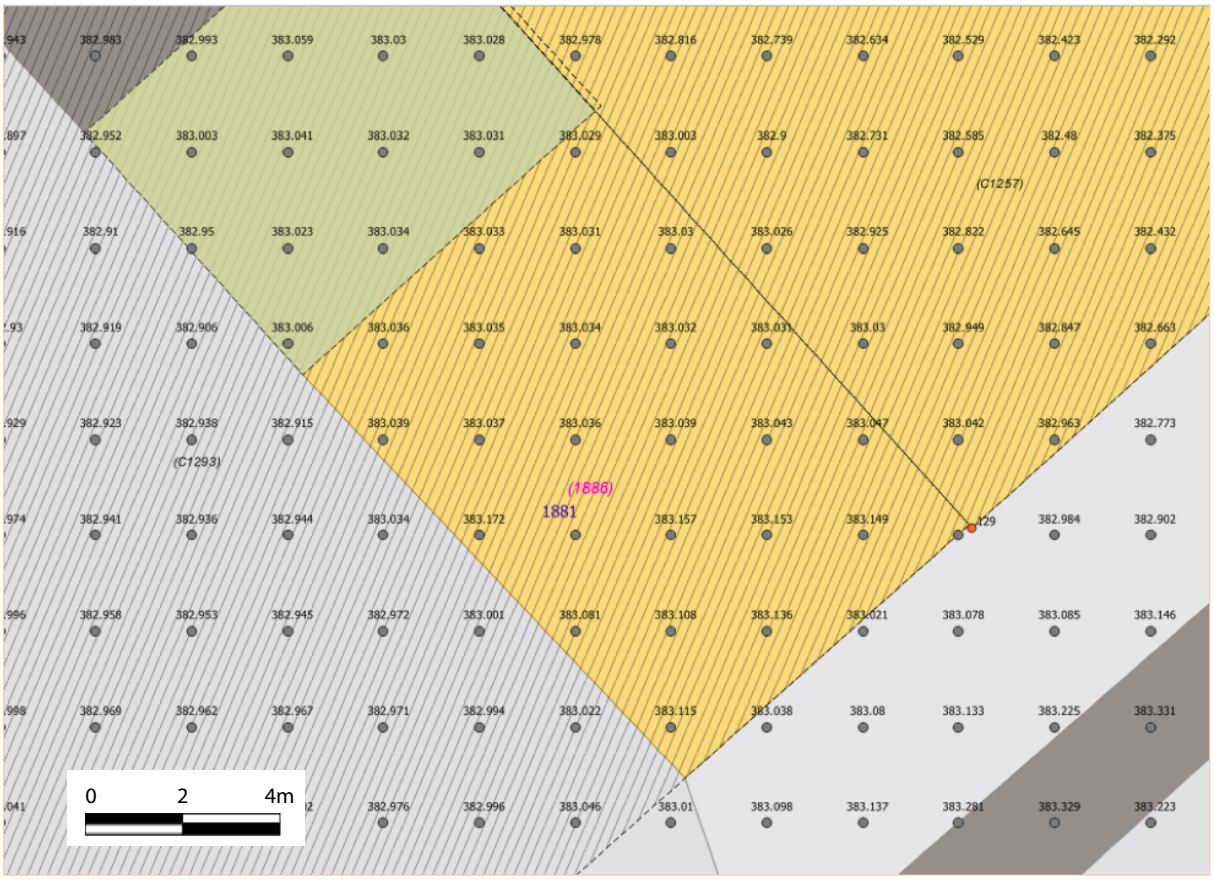


Figure 91 : Zoom sur la grille de points, avec affichage des altitudes (QGis).

4.3 Calcul des altitudes du MNS via les grilles de points MNT et MNA

4.3.1 But du projet

Le but de ce projet est de récupérer les valeurs des grilles de points calculées pour le MNT et le MNA, afin de calculer les altitudes du MNS (en additionnant les valeurs des deux autres modèles numériques).

L'idée est d'avoir un attribut commun entre les deux tuiles MNT et MNA. L'attribut « location » permet de faire le lien entre les deux types de modèles numériques, mais n'est pas assez spécifique, car chaque tuile contient 250'000 points, qui ont tous la même « location ». Les attributs *column* et *row* peuvent faire l'affaire ; chaque point possède un numéro de colonne et de ligne qui lui est propre. Il suffit de faire la concaténation de ces deux valeurs pour avoir un identifiant unique pour chaque point. Le souci est que deux tuiles MNA et MNT avec la même « location » (exemple 24961116), n'ont pas les mêmes valeurs de colonne et ligne pour les mêmes points (Figure 92 a et b). Il faut par conséquent créer un attribut unique à chaque point d'une même tuile, qui doit être le même pour le MNT et le MNA.

a

	fme_basena	value	_column	_row	location
1	MNA_HAUTEUR_2019	20.174011	1499	2999	24961116
2	MNA_HAUTEUR_2019	20.837006	1499	2998	24961116
3	MNA_HAUTEUR_2019	18.839996	1499	2997	24961116
4	MNA_HAUTEUR_2019	26.519989	1499	2996	24961116
5	MNA_HAUTEUR_2019	26.269989	1499	2995	24961116
6	MNA_HAUTEUR_2019	23.546997	1499	2994	24961116

b

	fme_basena	value	_column	_row	location
1	MNA_TERRAIN_2019	389.08	999	3499	24961116
2	MNA_TERRAIN_2019	388.845	999	3498	24961116
3	MNA_TERRAIN_2019	388.867	999	3497	24961116
4	MNA_TERRAIN_2019	388.707	999	3496	24961116
5	MNA_TERRAIN_2019	388.629	999	3495	24961116
6	MNA_TERRAIN_2019	388.615	999	3494	24961116

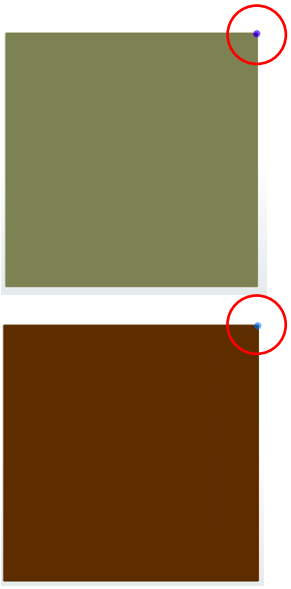


Figure 92 : Attributs du modèle numérique avec les valeurs de lignes et colonnes : **a)** MNA et **b)** MNT.

4.3.2 Marche à suivre

Nous disposons des grilles des points séparés de 2 mètres pour le MNT et le MNA (obtenus précédemment), et nous souhaitons additionner les valeurs de leurs hauteurs pour obtenir la valeur du MNS (pour chaque point de la grille).

Tout d'abord, à partir du MNT ou du MNA, nous créons un shapefile contenant une cinquantaine de tuiles. Il faut charger le même nombre de tuiles dans les deux développements (pour le MNT et pour le MNA) dans le ***TestFilter***. Nous n'en prenons pas plus car cela surcharge le disque C de l'ordinateur. Le développement est le même que précédemment, qui a permis la création de grilles (Figure 93).

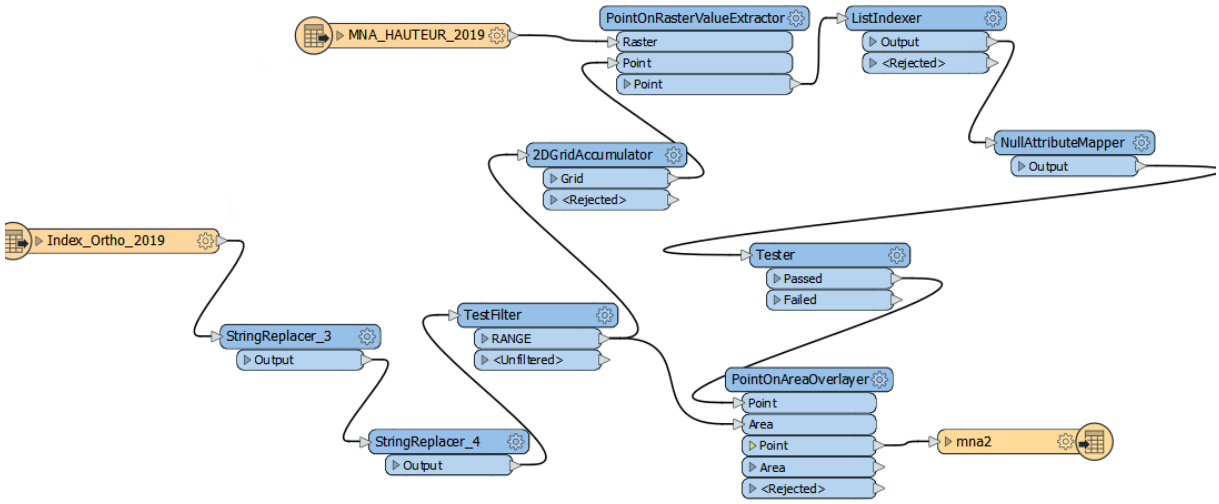


Figure 93 : Développement FME pour générer un shapefile contenant une cinquantaine de tuiles.

Les deux shapefiles en sortie (mna1 et mnt1) vont être intégrés dans un nouveau développement, qui permettra l'addition des valeurs d'altitude (Figure 94).

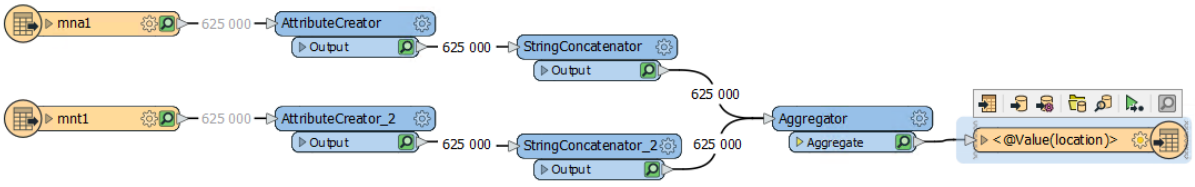


Figure 94 : Développement FME pour récupérer les valeurs d'altitude du MNT et MNA, afin de créer une nouvelle grille MNS.

Nous leur ajoutons un *transformer* qui permet de créer un identifiant unique pour chaque point (***AttributeCreator***). Nous appelons cet attribut « id », et ajoutons l'expression @Count(@CurrentAttribute()) dans la case *Attribute Value*, afin d'énumérer les entités, à partir de zéro (Figure 95).

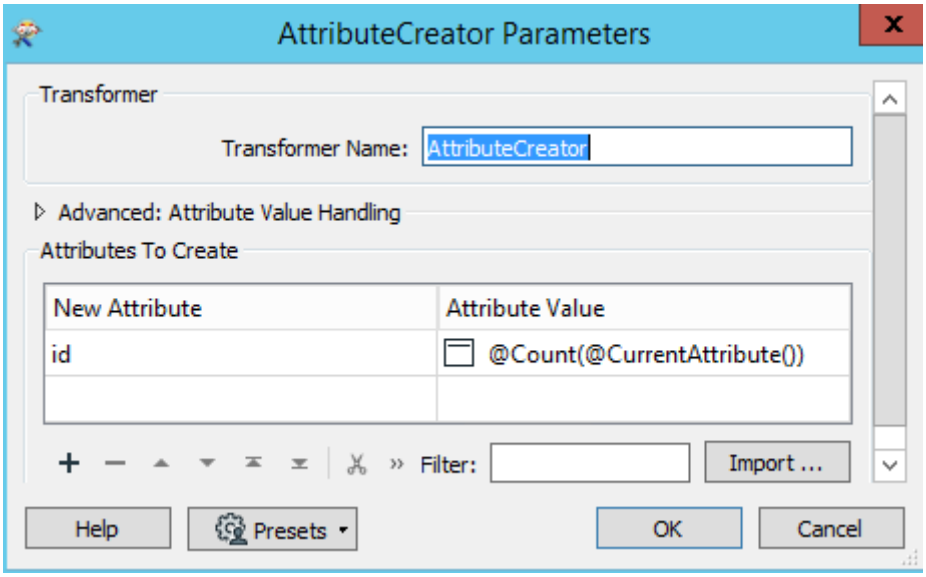


Figure 95 : Paramètres du transformer AttributeCreator, afin d'attribuer un id aux différents points de la grille.

	id	fme_basena	value	_column	_row	location
1	0	MNA_TERRAIN...	389.08	999	3499	24961116
2	1	MNA_TERRAIN...	388.845	999	3498	24961116
3	2	MNA_TERRAIN...	388.867	999	3497	24961116
4	3	MNA_TERRAIN...	388.707	999	3496	24961116
5	4	MNA_TERRAIN...	388.629	999	3495	24961116
6	5	MNA_TERRAIN...	388.615	999	3494	24961116
7	6	MNA_TERRAIN...	388.676	999	3493	24961116
8	7	MNA_TERRAIN...	388.865	999	3492	24961116
9	8	MNA_TERRAIN...	389.369	999	3491	24961116
10	9	MNA_TERRAIN...	389.574	999	3490	24961116
11	10	MNA_TERRAIN...	389.707	999	3489	24961116
12	11	MNA_TERRAIN...	389.791	999	3488	24961116
13	12	MNA_TERRAIN...	389.914	999	3487	24961116

Figure 96 : Résultat du transformer AttributeCreator, avec le nouvel attribut créé (id).

Ensuite, un ***StringConcatenator*** va nous permettre de créer un nouvel attribut (« _result »), qui sera la concaténation de l'identifiant unique (id) avec l'attribut « location », qui lui-même est la concaténation des coordonnées X et Y. Ce *transformer* permet de créer des chaînes de caractères, avec soit des attributs, des constantes (par exemple une virgule ou un point), des retours à la ligne etc.

Le résultat pour un point sera par exemple : 4, 24961116, et cela va être l'attribut commun entre deux tuiles (une MNT et une MNA) ayant les mêmes coordonnées. Cet attribut commun va nous permettre d'additionner leurs valeurs de hauteur.

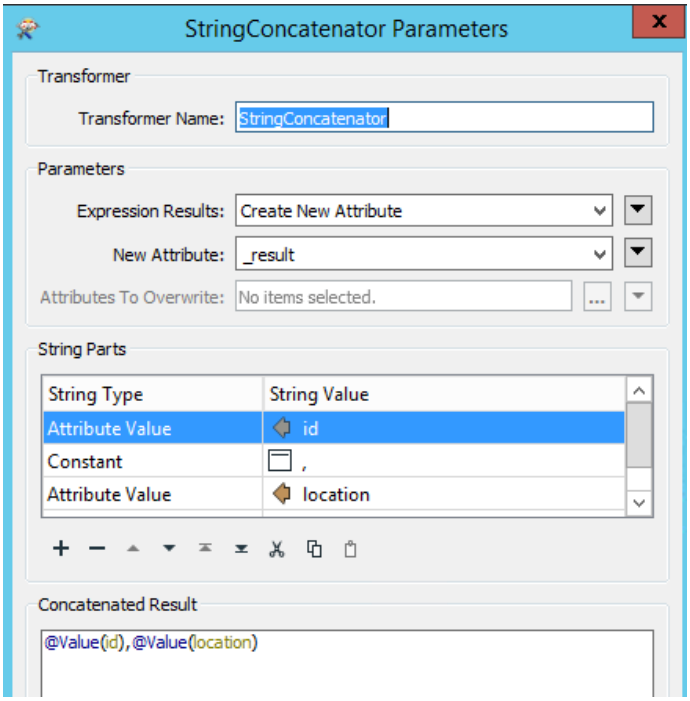


Figure 97 : Paramètres du transformer StringConcatenator, afin de concaténer les attributs id et location.

Un ***Aggregator*** va nous permettre d'additionner les valeurs, en groupant par le nouvel attribut créé (« _result ») et en additionnant l'attribut « value » (qui est l'altitude z).

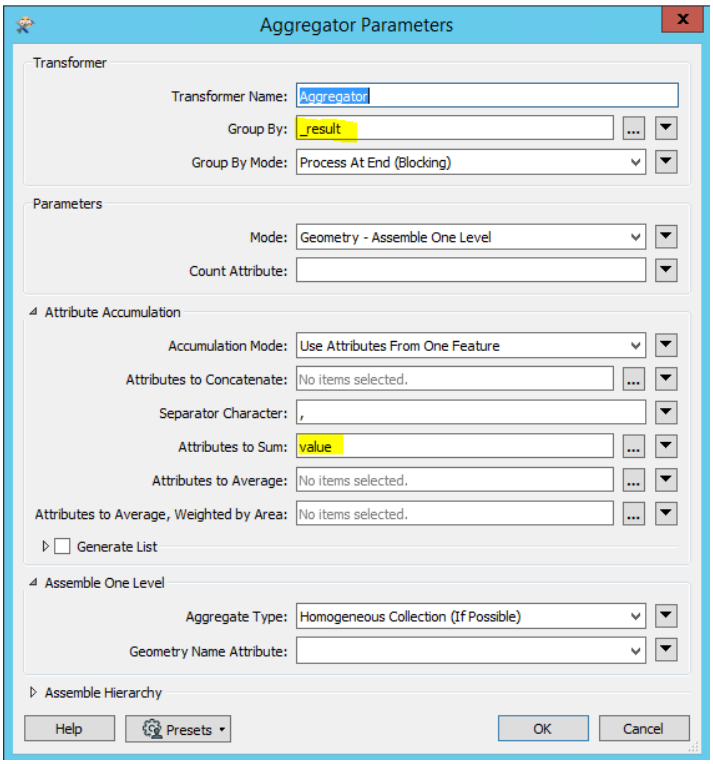


Figure 98 : Paramètres du transformer Aggregator. Groupement en fonction du résultat de la concaténation précédente (_result) et somme de l'attribut value.

Au final, nous obtenons des tuiles MNS, issues de l'addition des points du MNT et MNA. Le shapefile en sortie pourra à nouveau être chargé sur QGis via une action de type Python.

4.4 Création d'une grille de points sur un nuage de point LAS, en filtrant les points par catégorie

Les nuages de point LAS sont acquis par un lidar, qui projette un rayon laser en direction d'une surface sur terre, afin de mesurer la distance entre le centre d'émission et le point d'impact sur la surface de l'objet. Puisqu'on connait la direction d'émission du rayon laser et qu'on a obtenu la distance, on peut déterminer les coordonnées 3D du point relevé (www.leddartech.com, https://pro.arcgis.com). Le lidar opère par balayage, ce qui permet de mesurer un ensemble de points 3D dans l'espace environnant (nuage de points). Les données sont alors traitées pour produire un fichier de coordonnées (x, y, z). Le traitement des mesures et le filtrage des résultats permettent de distinguer une altitude de terrain, le MNT, et une altitude correspondant au sommet des objets (toits, arbres, etc.), le MNS (www.ne.ch). Ces modèles numériques sont sous forme de semis de points irrégulier, de grille de points à maille carrée ou triangulée (www.altoa.org).

4.4.1 But du projet

L'idée est de créer une grille de points sur un LAS, et de classer les points par catégories, en l'occurrence buildings, végétation et sol, afin d'avoir une interprétation plus détaillée de la donnée, qui ne sera pas uniquement un nuage de points. Ce développement n'a pas abouti à un résultat convaincant, mais la marche à suivre demeure pertinente.

4.4.2 Marche à suivre

Transformers

Point Cloud Filter
Point Cloud Coercer
CoordinateExtractor
2DGridAccumulator
NeighborFinder
PointCloudThinner

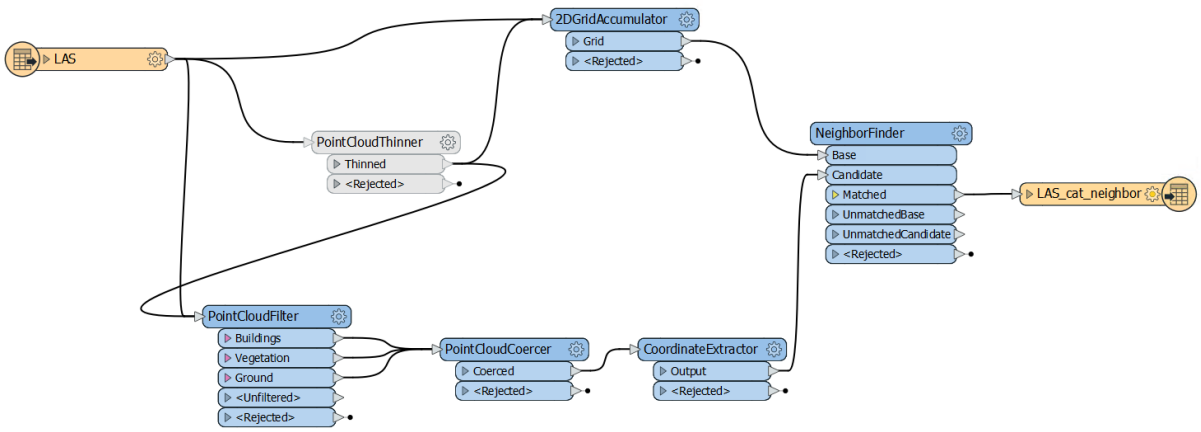


Figure 99 : Développement FME sur le nuage de points Lidar, afin d'obtenir une grille de points, catégorisés en fonction de trois classes (buildings, végétation et sol).

Pour catégoriser les points du LAS, nous appliquons un ***PointCloudFilter***. Ce *transformer* reçoit les caractéristiques du nuage de points et les divise en nuages de points individuels, sur la base de l'évaluation d'expressions définies. Dans ces expressions, nous créons une classification avec un numéro de classe bien particulier, propre à chaque catégorie (Figure 100).

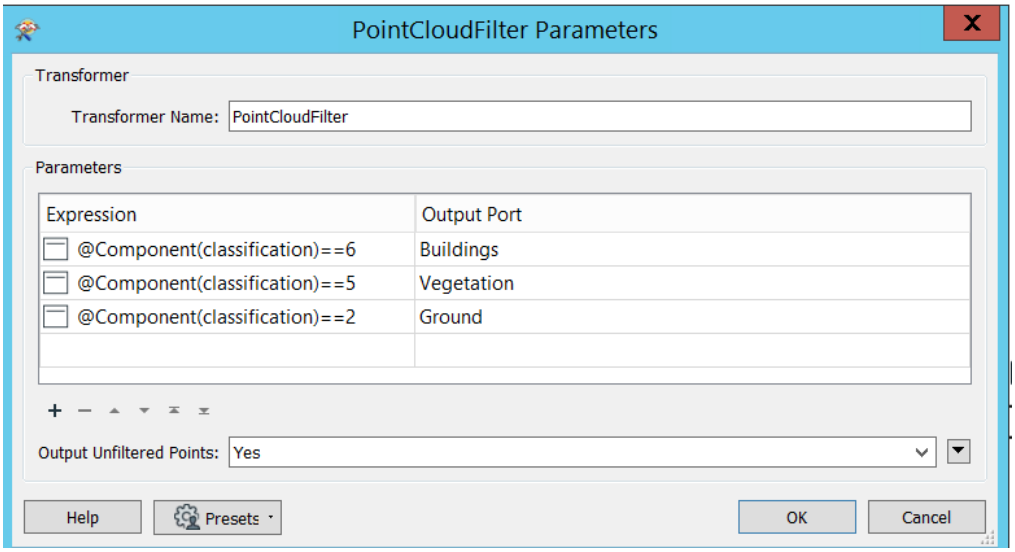


Figure 100 : Paramètres du transformer PointCloudFilter, avec les trois catégories définies.

Puis, nous appliquons un ***PointCloudCoercer***, afin de convertir le nuage de point (las) en points (shp). Ce *transformer* contraint les géométries des nuages de points à devenir des points ou des multipoints et il peut être utilisé pour convertir un nuage de points dans un format qui ne les supporte pas. Puis, il faut appliquer un ***CoordinateExtractor*** à ces points, afin d'en extraire l'altitude z (Figure 101).

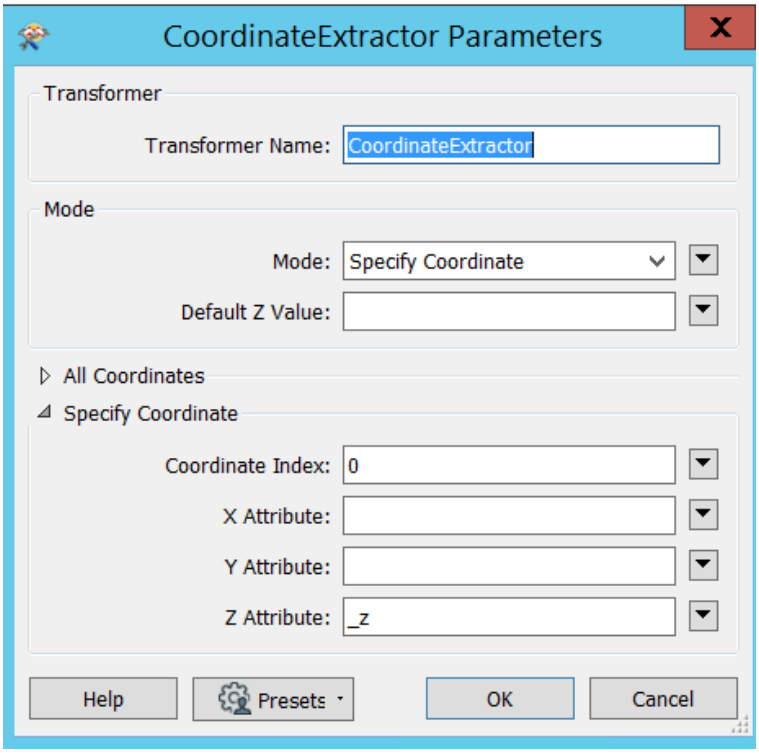


Figure 101 : Paramètres du transformer CoordinateExtractor, afin d'extraire la hauteur (z) des points.

En parallèle, nous appliquons une grille au fichier LAS (***2DGridAccumulator***), qui va nous permettre, comme pour les développements précédents, de définir un espacement entre les points auxquels on va attribuer une hauteur.

Finalement, un ***NeighborFinder*** va trouver les points les plus proches, à une distance prédéfinie (ici 0.5m). Il y aura des points *matched* et *unmatched*, et ce sont les *matched* qui nous intéressent, car ce sont les points qui ont trouvé un voisin à moins de 0.5 m (Figure 102). Un nouvel attribut « _distance » va être créé, contenant la valeur de la distance du point le plus proche. Ce *transformer* est nécessaire, car les nuages de points contiennent des points disposés aléatoirement entre eux, et il se peut que l'emplacement que nous avons défini par notre grille ne touche aucun point, c'est pourquoi nous récupérons le point le plus proche. Cette démarche est forcément beaucoup plus imprécise, en comparaison aux développements précédents sur les modèles numériques, qui possèdent une matrice de points beaucoup plus homogène et organisée.

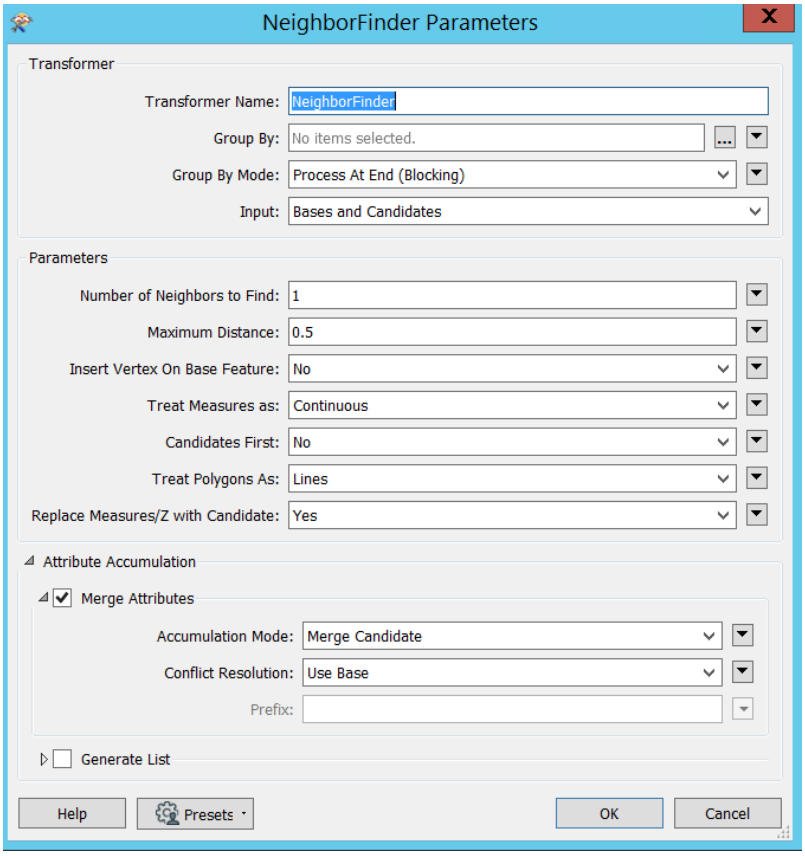


Figure 102 : Paramètres du transformer NeighborFinder, permettant de récupérer le point le plus proche, en fonction de la distance limite que l'on impose (ici 0.5 mètres).

Le *writer* est sous forme de shapefile, et il faut s'assurer de bien rentrer les champs « _z », « _distance » et « classification » dans la table attributaire du shapefile sortant, ainsi que de préciser shapefile_point dans la géométrie.

NB : Si l'on souhaite appliquer des *transformers* sur des nuages de points, il est préférable d'appliquer un ***PointCloudThinner***, qui va retirer des points à un certain intervalle choisi, car ce sont des fichiers lourds qui contiennent beaucoup de points. Lorsque le développement est fonctionnel, on peut l'appliquer à l'entièreté du LAS.

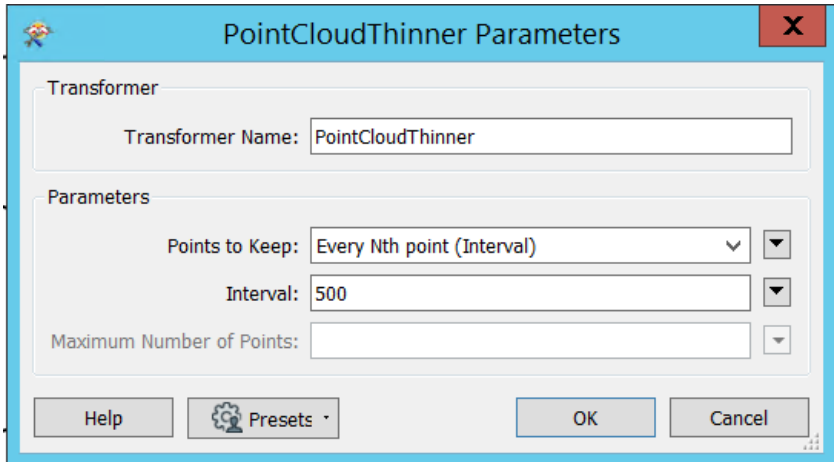


Figure 103 : Paramètres du transformer PointCloudThinner, permettant de réduire la densité de points, en fixant un intervalle de 500.

4.5 Projet de modification des coordonnées GPS des devis/dossiers

4.5.1 But du projet

Le but de ce projet est d'attribuer les coordonnées adéquates aux dossiers et devis mal géo-référencés sur le logiciel de gestion interne du bureau (Spadice), en les fusionnant avec des couches et/ou bases de données qui sont géo-référencés, afin de faire apparaître les bonnes coordonnées dans leur table attributaire. Les devis/dossiers ont été soit :

- Mal géo-référencés (hors Suisse, mauvais système de coordonnées géographiques ou coordonnées 0) ou mal notés.

- Géo-référencés sur les coordonnées des anciens bureaux (avant la fusion des deux bureaux Haller et Wasser), mais pas sur leur propre centroïde.

4.5.2 Méthode

Toutes les étapes se feront sur FME, à l'aide de différents *transformers*. Nous avons tout d'abord importé la base de données Access (mdb) de Spadice, contenant les dossiers et les devis Haller-Wasser. Nous avons également importé la base de données des archives Wasser (mdb), qui contient des dossiers et devis géo-référencés. Puis nous avons ajouté les couches des adresses, des parcelles et des parcelles historiques provenant du SITG, qui elles aussi sont géo-référencées.

4.5.3 Marche à suivre

Nous avons créé deux chemins de développement différents, un pour les devis et un pour les dossiers Spadice, en opérant de la même façon.

Transformers

Attribute Creator
Attribute Renamer
Attribute Reprojector
Attribute Validator
Center Point Replacer

Coordinate Extractor
Feature Merger
String Concatenator
String Replacer
Tester
Test Filter

4.5.4a Devis/dossiers liés à la géodatabase de Spadice

Séparation données Suisse – hors Suisse

Nous avons appliqué un ***TestFilter*** pour séparer les coordonnées en Suisse et hors Suisse. Les données hors Suisse sont celles en dehors du cadre que nous avons créé autour de la Suisse sur QGis.

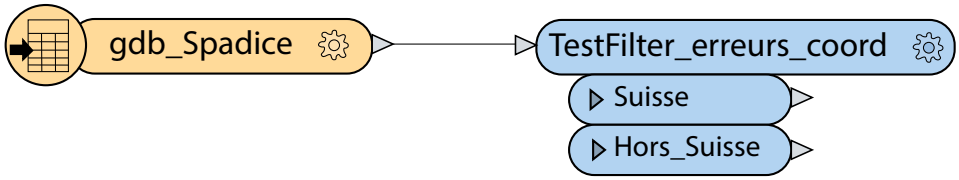


Figure 104 : TestFilter pour séparer les coordonnées Suisse et hors Suisse, de la base de données Spadice.

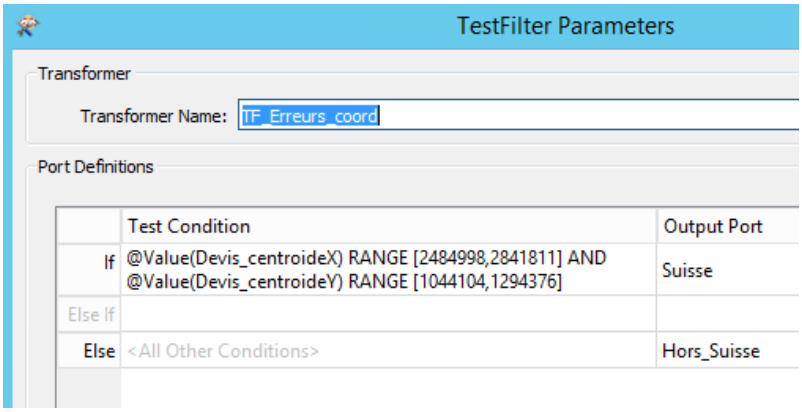


Figure 105 : Paramètres du TestFilter, avec l'emprise définie sur QGis autour de la Suisse (RANGE).

Dans les coordonnées en Suisse, nous avons séparé les données par anciens bureaux Haller et Wasser, soit l'emprise dans les alentours de la rue du Lièvre (anciens bureaux Haller) et de la Fontenette (anciens bureaux Wasser). Le champ *unfiltered* correspond à toutes les autres données de Spadice sur le canton.

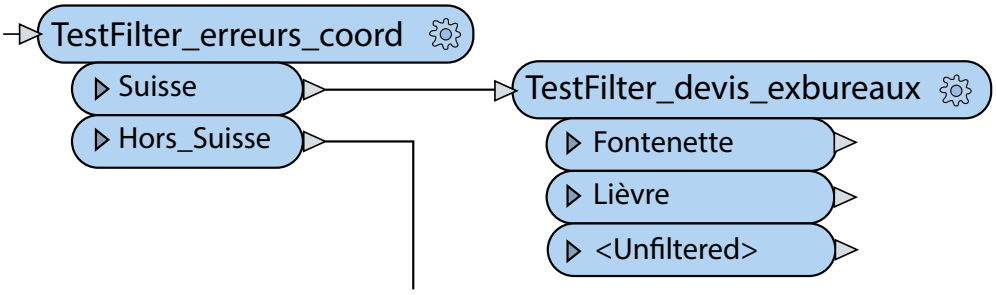


Figure 106 : TestFilter pour séparer les données des deux anciens bureaux (rue du Lièvre et rue de la Fontenette) du reste (unfiltered) des données.

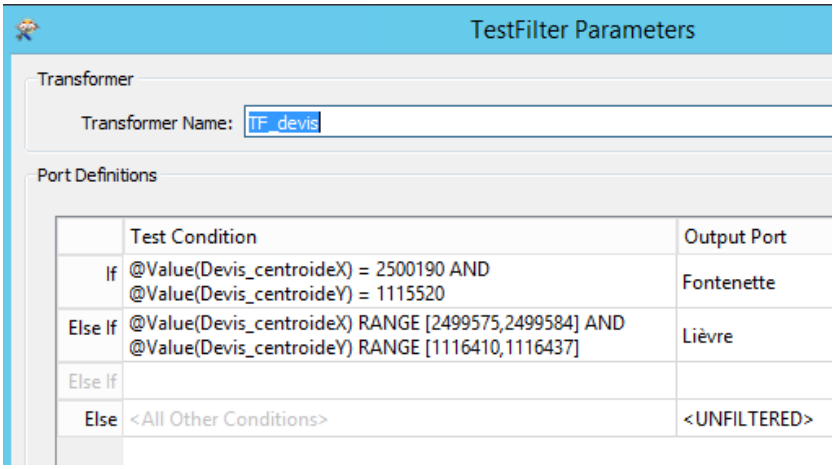


Figure 107 : Paramètres du TestFilter, avec l'emprise définie autour des deux anciens bureaux (RANGE).

En parallèle, concernant les coordonnées hors Suisse, nous avons ajouté un ***TestFilter*** pour cibler les données en MN03 (Figure 108). Nous leur avons appliqué un ***AttributeReprojector*** pour les mettre en MN95 (Figure 109), puis un ***AttributeRenamer*** pour renommer les champs existants des coordonnées en X et Y (Figure 110). Finalement, un ***AttributeCreator*** permet d'ajouter un attribut à la couche en sortie, où l'on précise quelle méthode nous a permis de retracer les coordonnées (Figure 111).

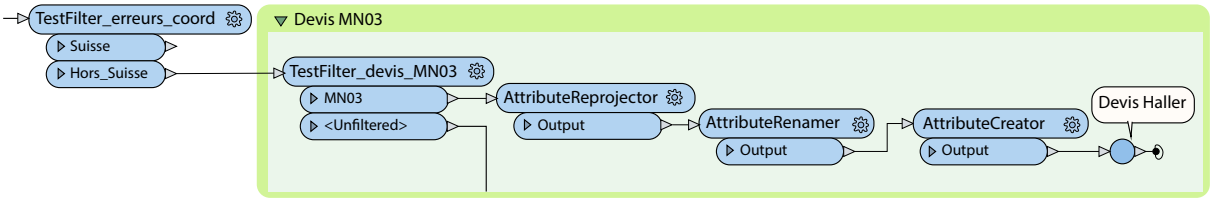


Figure 108 : Transformers appliqués aux données hors Suisse, afin de récupérer les coordonnées de celles-ci.

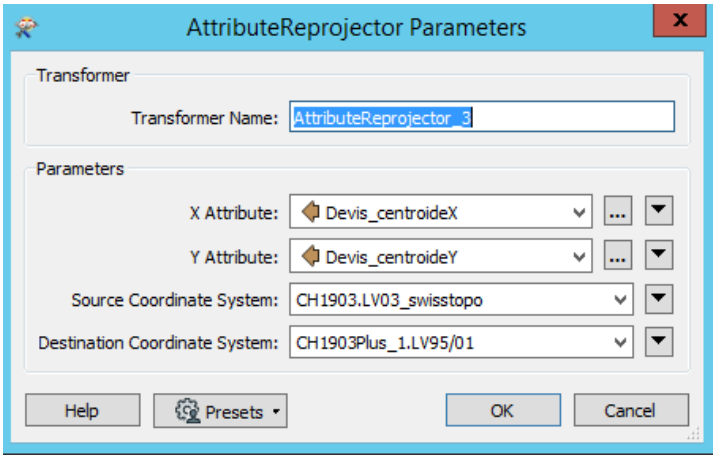


Figure 109 : Paramètres de l'AttributeReprojector, permettant de modifier le système de coordonnées (de MN03 à MN95).

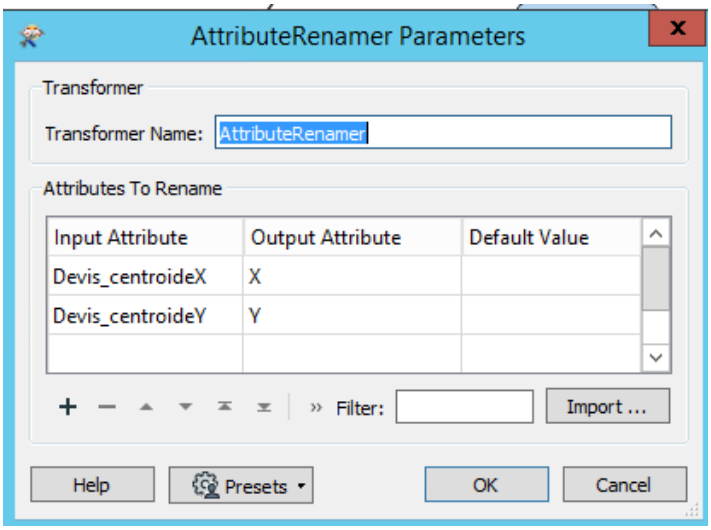


Figure 110 : Paramètres de l'AttributeRenamer, permettant de renommer les attributs des coordonnées en X et Y.

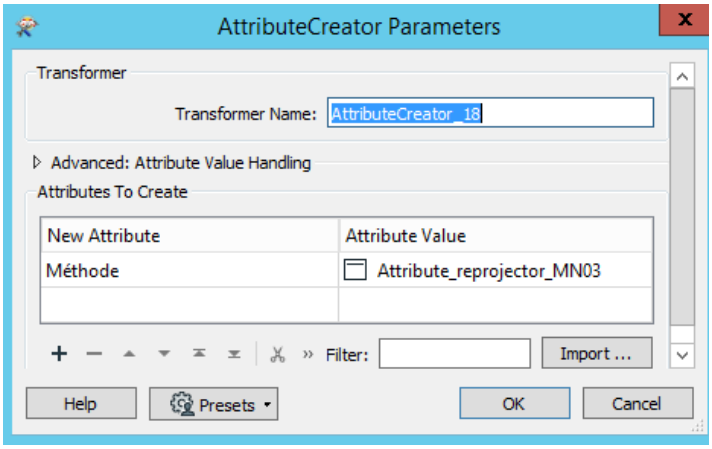


Figure 111 : Paramètres de l'AttributeCreator, afin de définir comment les coordonnées ont été retracées.

En revenant aux données en Suisse, nous avons ajouté un ***FeatureMerger***, qui s'applique aux devis ou dossiers de la rue du Lièvre (*Requestor*) et à la couche des communes dans Spadice (*Supplier*), en les liant par leur id commun, soit leur numéro de commune (Figures 112 et 113). Ce *transformer* va fusionner les attributs et/ou la géométrie d'un ensemble de caractéristiques sur un autre ensemble de caractéristiques, en se basant sur la correspondance des valeurs et des expressions des attributs. Il faut distinguer deux branches ; le *Requestor* et le *Supplier*. Le *requestor* est l'entité qui recevra de nouveaux attributs et/ou une nouvelle géométrie, et le *supplier* est l'entité qui les fournit (http://docs.safe.com). Ce qui nous intéresse est la branche *merged*, qui correspond aux entités fusionnées.

Le but de cette manœuvre est d'ajouter un attribut inexistant au *requestor*, en l'occurrence le nom de la commune. En effet, les devis ou dossiers contiennent le numéro de commune dans leurs attributs, mais pas le nom. Il est nécessaire qu'ils possèdent cet attribut, car par la suite, nous allons devoir lier les devis ou dossiers à des couches provenant du SITG, qui elles ne possèdent que le nom de la commune dans leurs attributs.

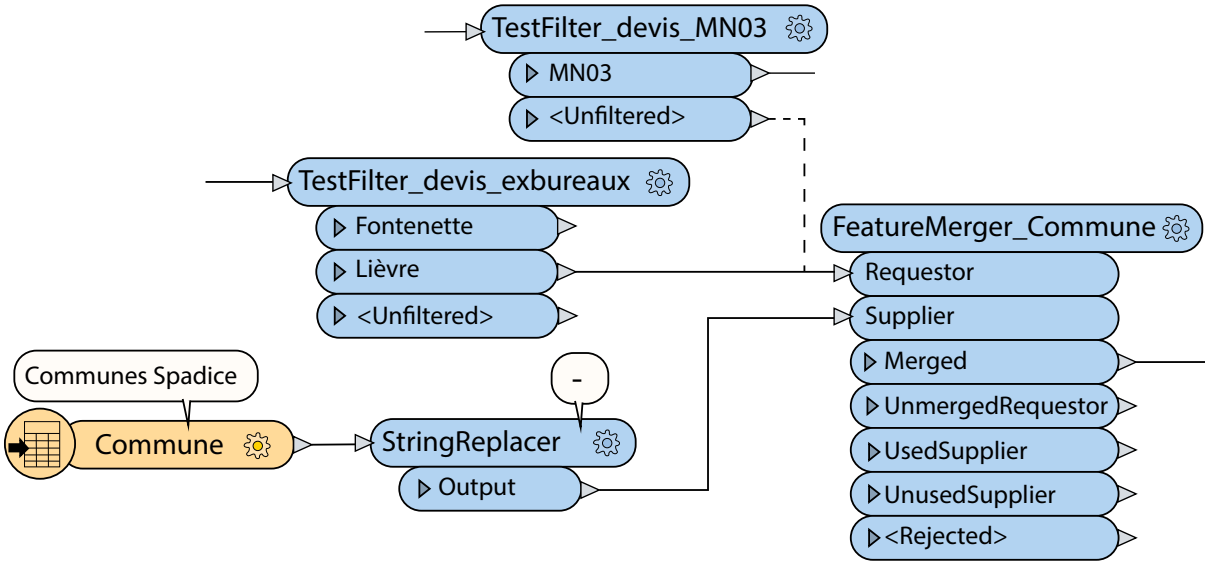


Figure 112 : FeatureMerger entre devis/dossiers et communes, afin d'attribuer les noms de communes aux devis/dossiers. Les traits-tillés indiquent qu'il y a d'autres transformers entre (voir figure 108).

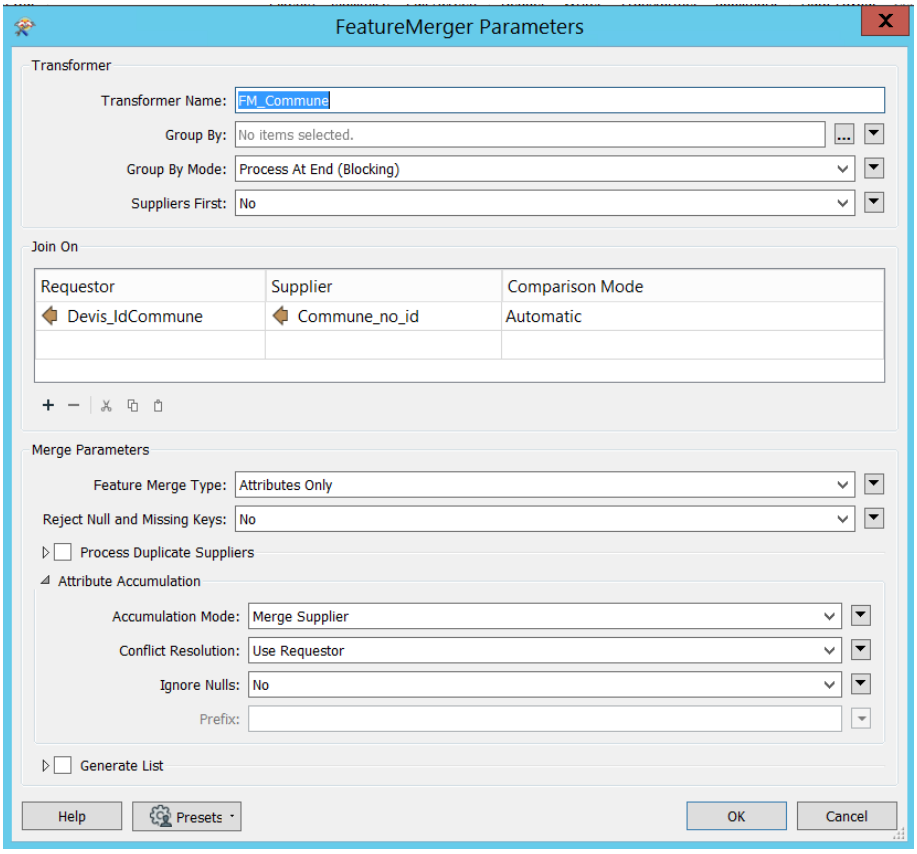


Figure 113 : Paramètres du transformer FeatureMerger. La fusion des attributs se fait en fonction de l'attribut commun, ici le numéro de commune.

Adresses et parcelles non nulles

Puis, à la branche *merged*, nous appliquons un ***TestFilter***, ayant pour condition de ne garder que les adresses et les parcelles non-nulles, dans le but de les lier par la suite avec la couche des adresses et des parcelles provenant du SITG. Les valeurs NULL ne peuvent pas être utilisées, car elles ne

possèdent ni adresse, ni parcelle, il est donc impossible de retracer leurs coordonnées. Le filtre est simple : lorsque qu'il y a des valeurs nulles dans les champs adresses et parcelles, cela crée la branche « NULL », et tout le reste est considéré comme « non NULL ».

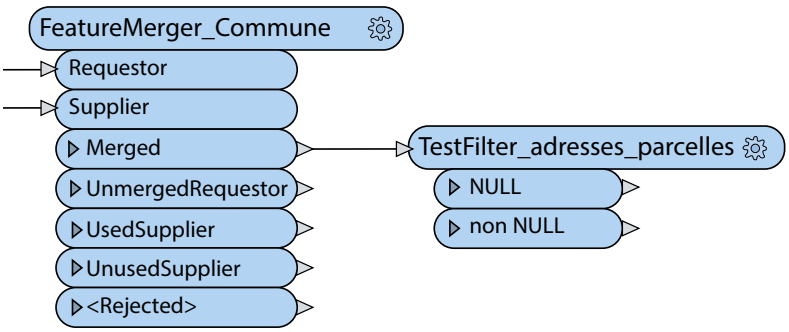


Figure 114 : TestFilter pour filtrer les adresses et les parcelles non nulles.

Aux données non nulles, nous avons ajouté toute une série de ***StringReplacer*****,** qui permettent de modifier des éléments présents dans les attributs, tels que des points, des virgules, des tirets et d'autres types de symboles ou texte afin d'avoir des informations similaires dans notre couche et celle du SITG (Figure 115).

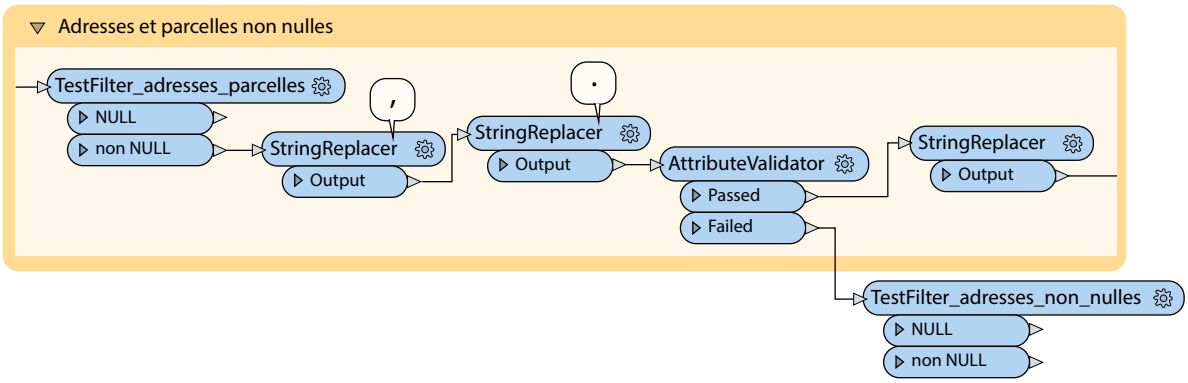


Figure 115 : Développement pour les adresses et parcelles non nulles.

Comme certaines adresses de la table de Spadice ne sont pas affichées dans le même ordre que les adresses du SITG, nous avons utilisé un ***AttributeValidator*** sur la branche « non NULL » du « TestFilter_adresses_parcelles » pour séparer les adresses qui ont le numéro au début, de celles qui l'ont à la fin. Ce *transformer* renvoie deux possibilités, *passed* et *failed* (Figure 116). En tant que *Validation Rule*, nous avons opté pour *Contains Regex*, où *Regex* signifie *Regular Expression*. En notant ^[0-9], on indique au *transformer* qu'il ne doit considérer que les cas qui commencent par des chiffres (Figure 117).

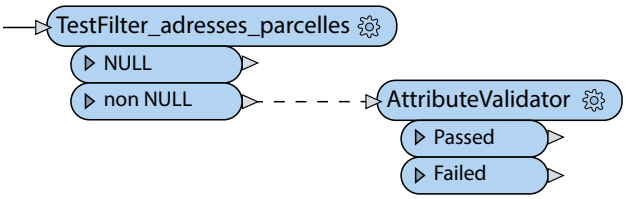


Figure 116 : AttributeValidator sur les valeurs non nulles des adresses et parcelles. Les trait-tillés indiquent qu'il y a d'autres transformers entre le TestFilter et l'AttributeValidator (voir figure 115).

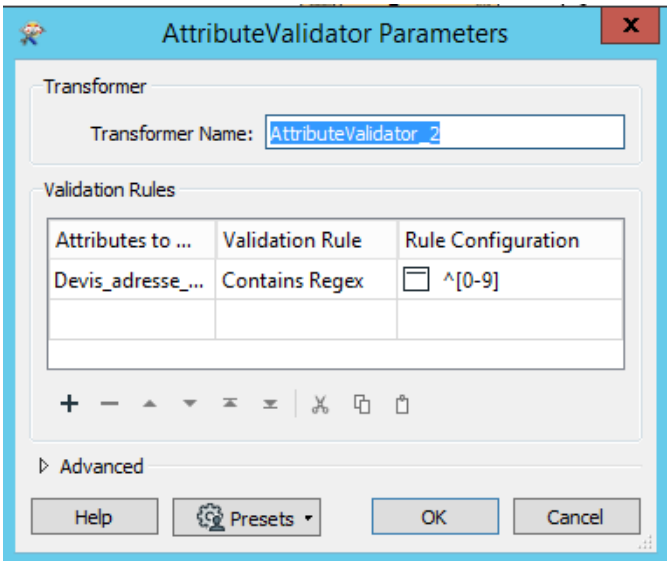


Figure 117 : Paramètres de l'AttributeValidator, avec la Regular expression imposée.

En parallèle, nous avons lié un ***CoordinateExtractor*** aux adresses SITG, pour que les coordonnées X et Y soient extraites et apparaissent dans les attributs de la couche. Puis, nous avons effectué un ***StringConcatenator*** sur les adresses SITG pour mettre les informations dans l'ordre qui nous intéresse (Figure 118).

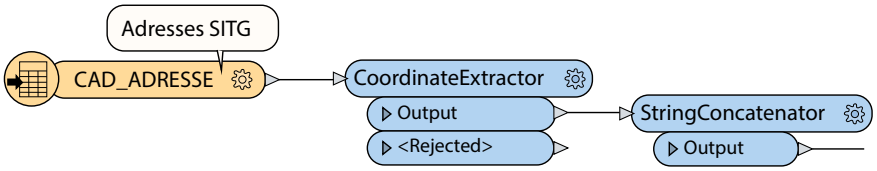


Figure 118 : Transformers appliqués à la couche des adresses provenant du SITG.

En effet, cette couche contient plusieurs champs d'adresse, soit contenant l'adresse complète (ADRESSE), soit l'adresse découpée par entités (TYVOIE, LIANT, NOMVOI, NO_ADRESSE) (Figure 119). L'ordre est important, car si une adresse n'est pas complètement identique, le ***FeatureMerger*** ne fonctionne pas.

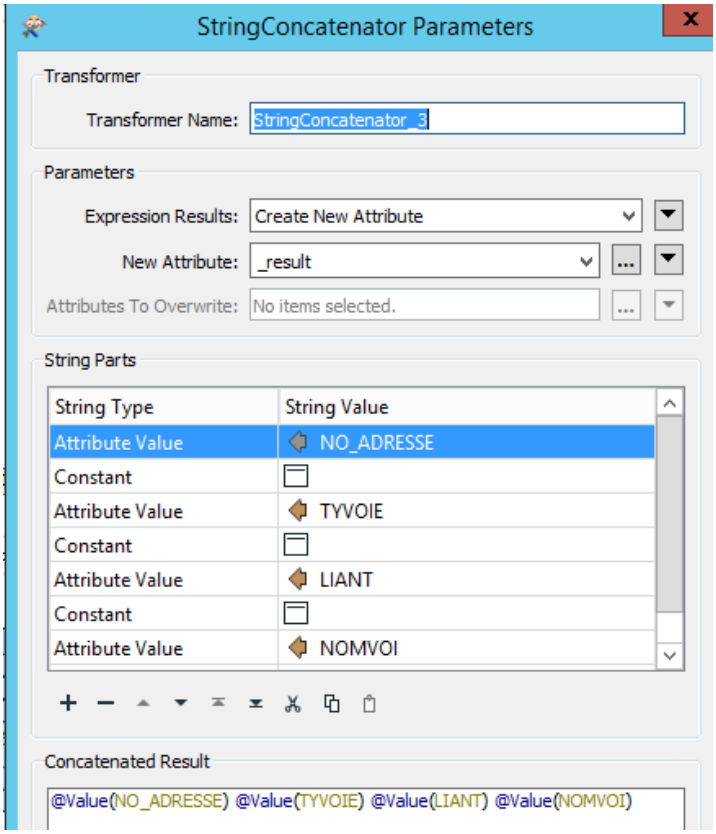


Figure 119 : Paramètres du StringConcatenator, dans le but de concaténer les attributs d'adresse dans l'ordre souhaité.

Finalement, nous avons lié les résultats des adresses et parcelles non nulles aux adresses du SITG, par le biais d'un ***FeatureMerger***, avec leur identifiant commun, *i.e.* l'adresse. Les données *merged* ont été sauvées dans un tableau Excel, alors que les *unmerged* subissent d'autres tests pour récupérer leurs coordonnées.

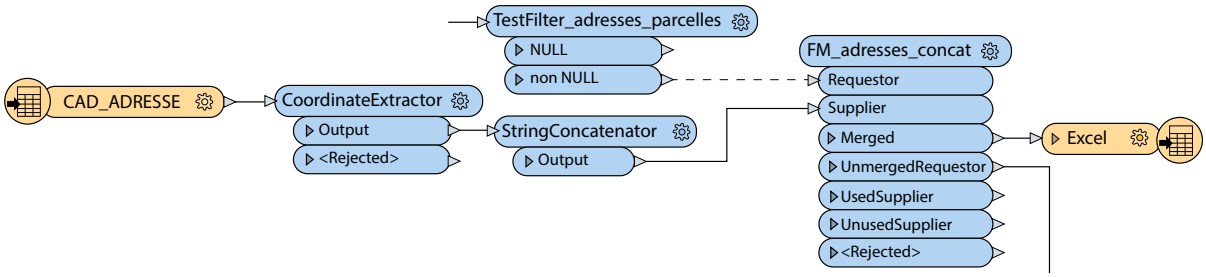


Figure 120 : FeatureMerger entre les adresses des devis/dossiers et les adresses SITG.

Aux données *failed* de l'***AttributeValidator*** (Figure 115), nous avons appliqué un ***TestFilter***, pour ne garder que les adresses non-nulles. Nous avons appliqué une série de ***StringReplacer*** et de ***Tester*** pour modifier certains caractères dans les adresses (*exemple* : remplacer av par avenue, ch par chemin etc.) (Figure 121).

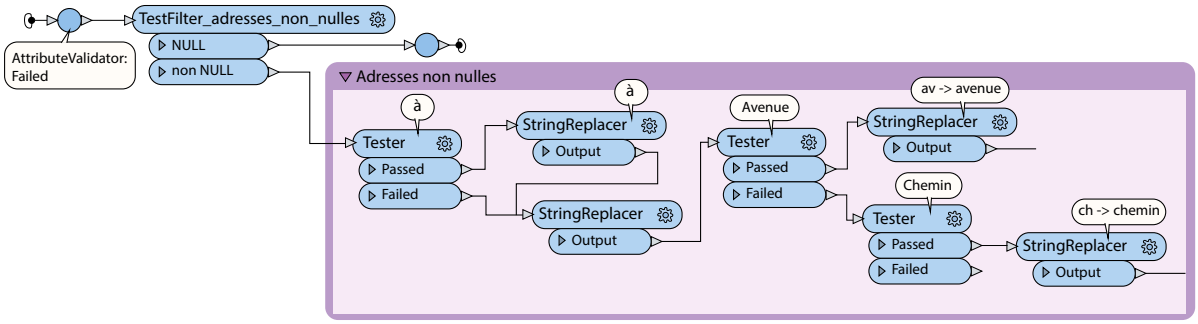


Figure 121 : Transformers appliqués aux données failed de l'AttributeValidator.

Nous avons à nouveau généré un ***FeatureMerger*** entre les adresses SITG et la sortie de nos divers tests sur les adresses. Les valeurs *merged* ont été liées au fichier Excel en sortie, alors que les *unmerged* ont à nouveau subi des *transformers* (Figure 122) ; un ***AttributeValidator***, dans le but de séparer les adresses contenant deux chiffres (*exemple* : 10-12 rue de Carouge). L'expression permettant de déceler ces adresses est la suivante : [0-9]-[0-9]. Cette *Rule configuration* va chercher dans la couches les cas où un chiffre de 0 à 9 est séparé d'un autre chiffre de 0 à 9 par un tiret (Figure 123).

Puis un ***AttributeSplitter*** va découper l'attribut en fonction du délimiteur (-) qu'on applique (Figure 124). Une liste va être générée, par conséquent il faut appliquer un ***ListIndexer***, pour identifier les éléments en question (Figure 125). Et finalement, on applique à nouveau un ***FeatureMerger***.

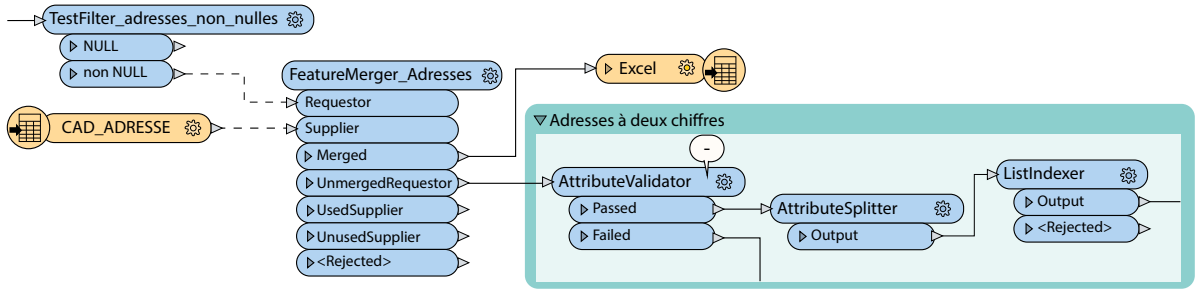


Figure 122 : Développement pour modifier les adresses à deux chiffres.

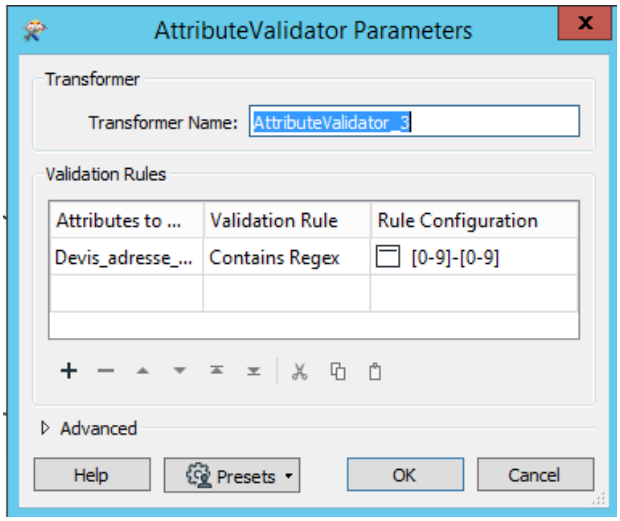


Figure 123 : Paramètres de l'AttributeValidator, avec la Regular expression imposée.

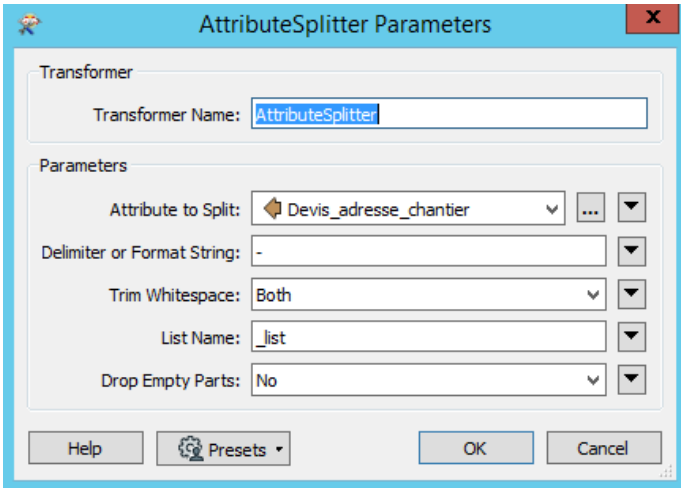


Figure 124 : Paramètres de l'AttributeSplitter, afin de scinder l'attribut en deux, grâce au délimiteur défini.

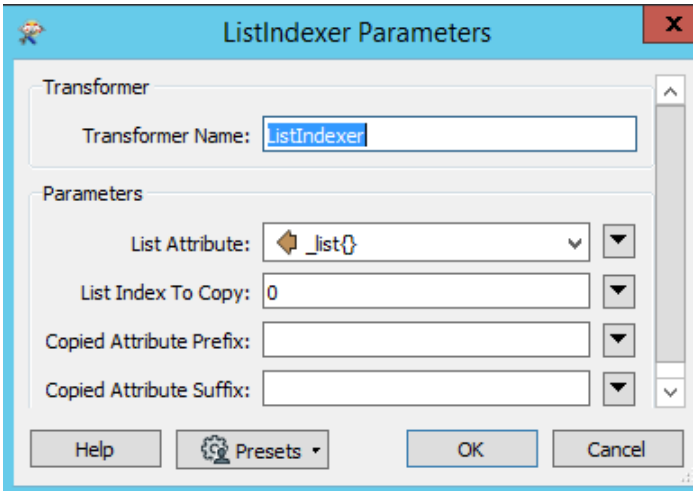


Figure 125 : Paramètres du ListIndexer.

Communes et parcelles

Le test suivant est celui des communes et des parcelles. Nous avons d'abord appliqué un ***TestFilter***, pour ne garder que les parcelles non-nulles qui découlent des adresses *unmerged* de tous nos ***FeatureMerger***.

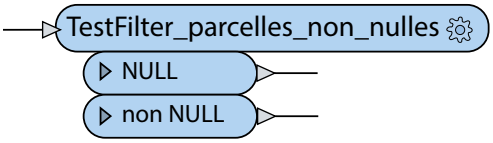


Figure 126 : TestFilter pour les parcelles non nulles.

Puis, nous avons ajouté un ***FeatureMerger***, en liant la couche des parcelles SITG (*Supplier*) aux parcelles non-nulles (*Requestor*), par leur commune et numéro de parcelle (Figure 127).

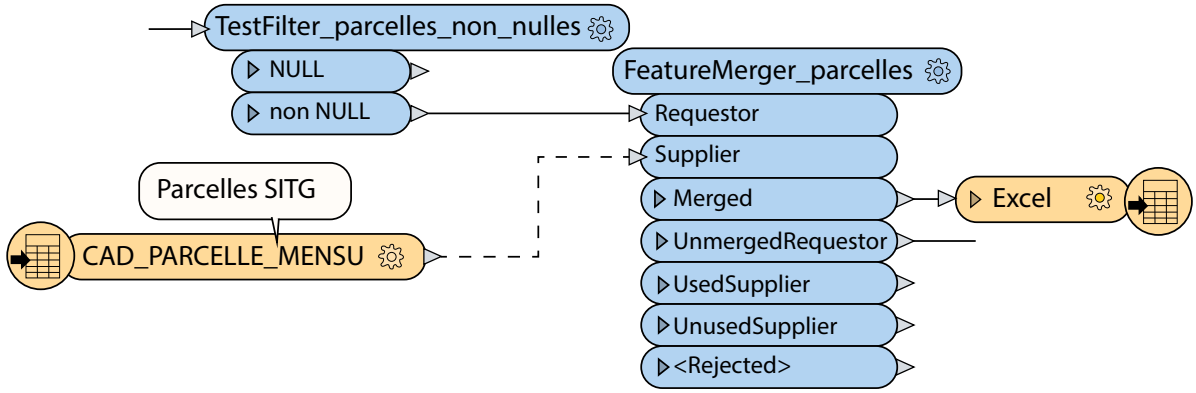


Figure 127 : FeatureMerger entre les parcelles non nulles et les parcelles provenant du SITG, ayant comme attribut commun les noms de communes et les numéros de parcelles. Les traits-tillés indiquent que d'autres transformers se trouvent entre la couche des parcelles du SITG et le FeatureMerger.

Communes et parcelles historiques

Les données *merged* ont été sauvées dans un tableau Excel, alors que les *unmerged* subissent un nouveau ***FeatureMerger***, cette fois-ci en utilisant la couche des parcelles historiques en tant que *supplier*. En effet, les parcelles subissent fréquemment des divisions ou des fusions, par conséquent leur numéro de parcelle est modifié et n'existe plus, c'est pourquoi nous faisons appel à la couche des parcelles historiques, qui contient tous les anciens numéros de parcelles.

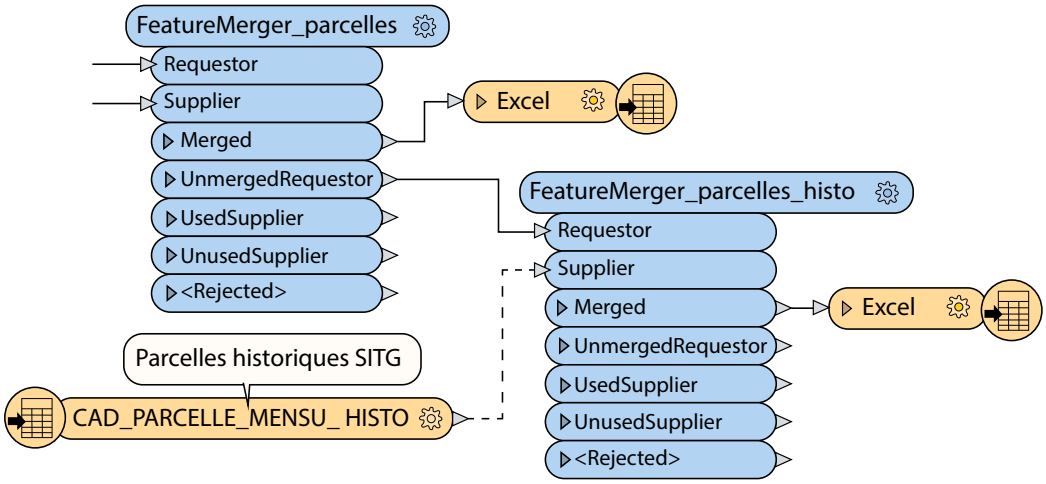


Figure 128 : FeatureMerger entre les parcelles historiques SITG et la branche unmerged du FeatureMerger précédent. A nouveau, l'attribut en commun est le nom des communes et les numéros de parcelles. Les traits-tillés indiquent que d'autres transformers se trouvent entre la couche des parcelles historiques du SITG et le FeatureMerger.

En parallèle, nous avons appliqué le transformer ***CenterPointReplacer*** sur les couche des parcelles et des parcelles historiques, dans le but de remplacer la géométrie de l'entité par un point qui sera forcément situé dans un endroit à l'intérieur de la zone d'entité (*mode : any inside point*). Puis un ***CoordinateExtractor*** pour obtenir les coordonnées GPS (Figure 129).

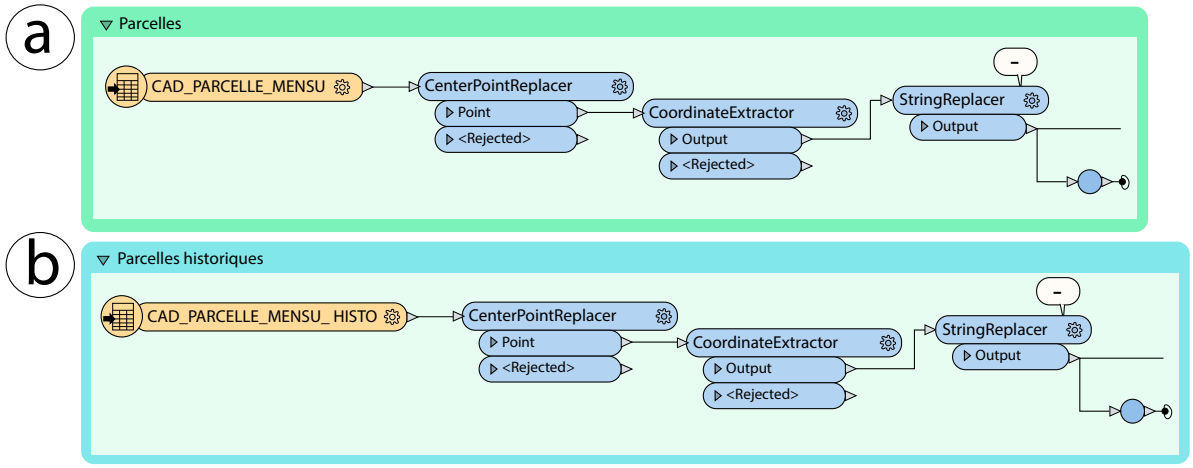


Figure 129 : Développement pour récupérer les coordonnées des **a)** parcelles et **b)** parcelles historiques.

4.5.4b Devis/dossiers liés à la géodatabase Wasser

Concernant les devis ou dossiers de la Fontenette (ex bureaux Wasser), les étapes sont différentes car les coordonnées peuvent être directement retracées grâce à la base de données « Archives » de Wasser. En tout premier lieu, nous séparons les devis des dossiers par le biais d'un ***TestFilter***. La branche « Reste » correspond aux numéros de dossiers, qui sont présents dans diverses colonnes. En effet, pour chaque type de prestation (levé, EDL, cadastration, etc.) il y a un attribut « dossier » différent. Nous les avons tous regroupés par le biais du ***TestFilter***, en disant que tout ce qui n'est pas un devis est un dossier.

Ensuite, il suffit de faire un ***CoordinateExtractor*** pour extraire les coordonnées GPS, puis un ***AttributeReprojector*** pour les mettre dans le même système de coordonnées que sur Spadice, c'est-à-dire en MN95 (Figure 130). Par ailleurs, sur la géodatabase Wasser, les devis ont le préfixe D, alors que sur Spadice ils sont préfixés par un 2. Un ***StringReplacer*** permet de faire cette modification, pour que les devis puissent être liés par leur numéro de devis commun.

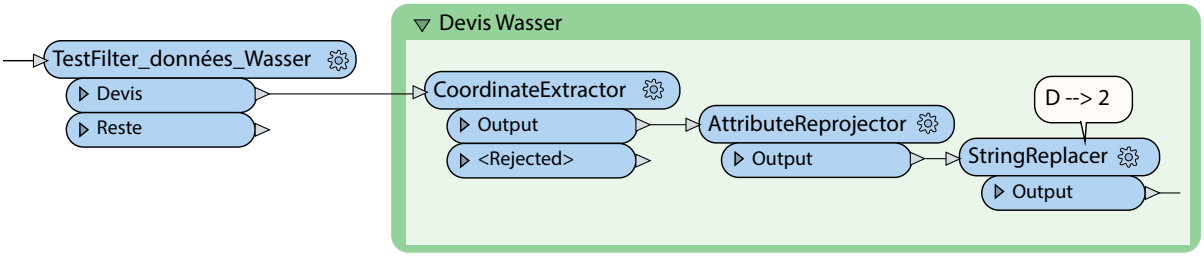


Figure 130 : Développement pour les devis Wasser, provenant d'une géodatabase MDB.

Finalement, il faut faire un ***FeatureMerger*** entre la base de données Archives et la chaîne Fontenette de Spadice pour obtenir les coordonnées des devis/dossiers des anciens bureaux Wasser. A nouveau, ces données seront sauvées dans la table Excel.

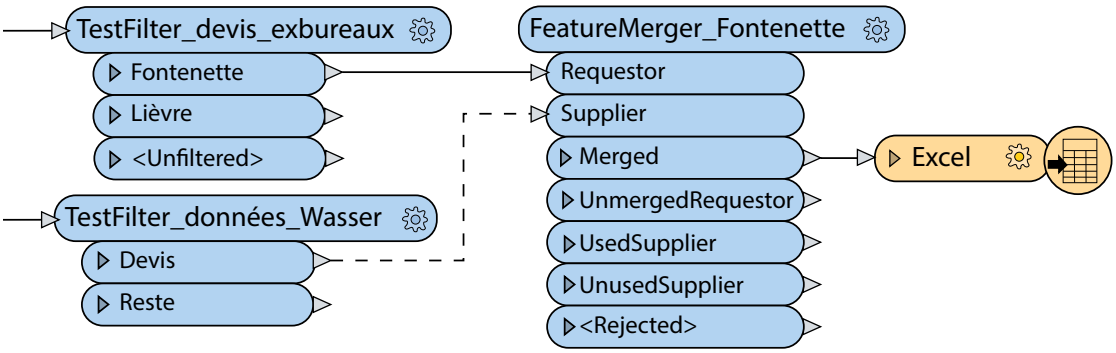


Figure 131 : FeatureMerger entre la base de données Wasser et les données de la base Spadice, pour les bureaux de la Fontenette (ex-Wasser).

NB : A chaque fois que l'on souhaite lier une sortie de *transformer* à la table Excel, nous appliquons un ***AttributeCreator***, en le nommant par la méthode utilisée pour récupérer les coordonnées, pour que l'on puisse savoir sur Excel d'où proviennent toutes ces coordonnées.

5. Projets Python-QGis

5.1 But du projet

L'objectif est la création d'un plugin sur QGis qui permet d'exporter les points sélectionnés, relatifs à une couche, au format csv. Les points à exporter sont essentiellement les points obtenus par les géomètres sur le terrain, et les points du cadastre. L'idée sera ensuite d'ajouter ce fichier de points aux appareils de terrain, notamment le théodolite, pour avoir toutes les données nécessaires à portée de main lors de sorties sur le terrain.

Un plugin similaire, développé par Michael Minn alias MMQGIS, permet d'exporter des points en csv. Cependant, il ne permet pas d'exporter la sélection de points, mais exporte toutes les entités de la couche. Nous nous inspirerons de son développement pour mener à bien le nôtre. Nous allons créer un plugin sur QGis, où il sera nécessaire de coder en Python, et nous en définirons l'interface via Qt Designer. La console de QGis nous permettra de nous entrainer en Python, afin de voir les résultats de morceaux de code en instantané. De plus, des bouts de code ont été intégrés dans Anaconda, afin de déceler des possibles erreurs de code. Une fois le code terminé, celui-ci a été enregistré dans les fichiers Python générés par le plugin, et le résultat est visible directement sur QGis.

5.2 Marche à suivre

En tout premier lieu, nous avons installé les plugins « Plugin Builder »  et « Plugin Reloader »  , dans les extensions de QGis. Le « Plugin Builder » permet de créer un plugin, en définissant plusieurs paramètres. Il est notamment possible de préciser où l'on souhaite que le plugin apparaisse dans l'interface de QGis (sous plugins, database, raster, vector ou web) (Figure 132).

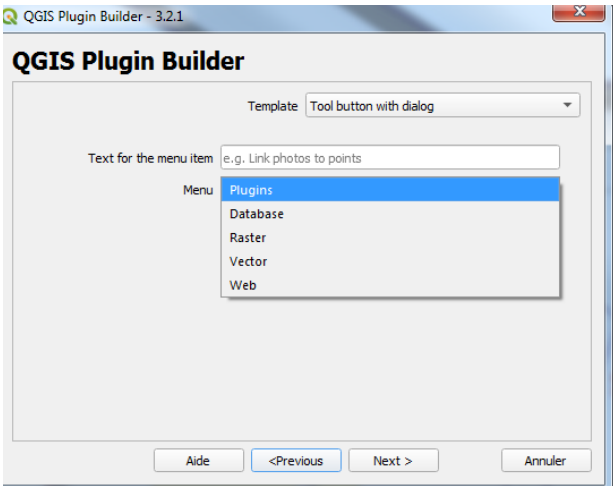


Figure 132 : Paramètres à définir lors de la création du plugin.

Le plugin va ensuite être sauvé sous :

C:\Users\nom_utilisateur\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins

Lorsque l'on clique sur *Generate*, un message s'affiche, concernant la compilation des ressources (Figure 133), qui sera réglé par la suite. Il faut ensuite fermer QGis, pour que le nouveau plugin se mette à jour lorsque l'on redémarre le logiciel.

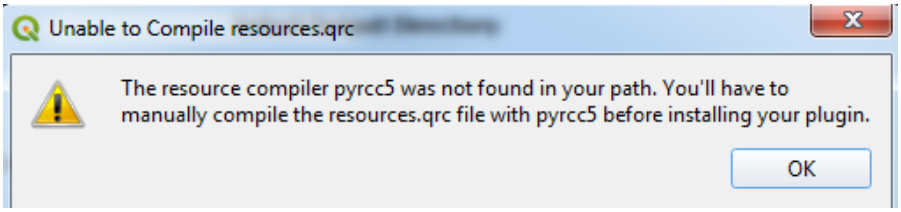


Figure 133 : Message d'avertissement pour la compilation des ressources.

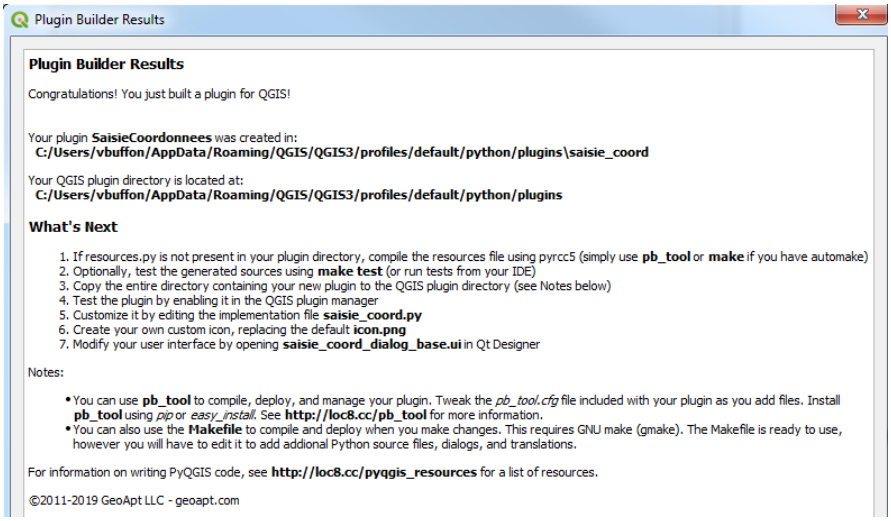


Figure 134 : Message lors de la création du plugin.

5.3 Dossier du plugin

Un dossier avec le nom du plugin attribué va se créer dans le même chemin où nous avons sauvé le plugin. Ce dossier comporte plusieurs fichiers Python (.py), des fichiers texte, ainsi qu'un fichier QT ui. Tous ces fichiers se créent automatiquement, à l'exception des fichiers ressources, qui sont créés par nos soins.

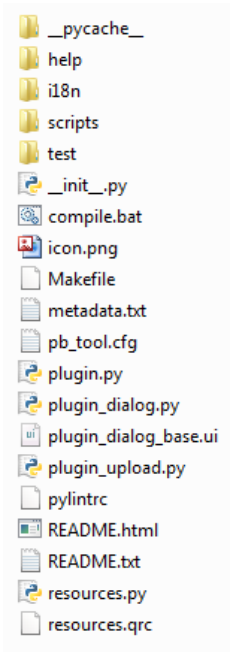


Figure 135 : Fichiers présents dans le dossier du plugin.

5.4 Générer un fichier resources.py

Pour générer un fichier resources.py, il faut créer un nouveau fichier texte brut dans le dossier du plugin et coller les commandes suivantes dans ce fichier :

```
@echo off
call "C:\Program Files\QGIS 3.10\bin\o4w_env.bat"
call "C:\Program Files\QGIS 3.10\bin\qt5_env.bat"
call "C:\Program Files\QGIS 3.10\bin\py3_env.bat"

@echo on
pyrcc5 -o resources.py resources.qrc
```

NB : vérifier que la version de QGis soit la bonne (ici : 3.10).

Puis, enregistrer ce fichier sous : compile.bat (fichier de commande Windows). Ceci va créer un fichier resources.py sous forme de Python file. Il suffit de double cliquer sur ce fichier pour l'exécuter.

5.5 QT Designer

Pour créer l'interface graphique du plugin, il faut ouvrir le fichier .ui avec Qt Creator ou Qt Designer. Ces logiciels permettent de concevoir et construire des interfaces graphiques (GUI), en personnalisant les fenêtres ou dialogues par l'ajout de Widgets Qt. Ces Widgets sont de toutes sortes ; *push buttons*, *combo box*, *labels* etc. (Figure 137). Sur Qt, les différents outils peuvent être renommés, mais dans le code Python il faut être attentif à faire référence au nom défini sur Qt, pour que Python fasse le lien avec les objets auxquels nous nous référons.

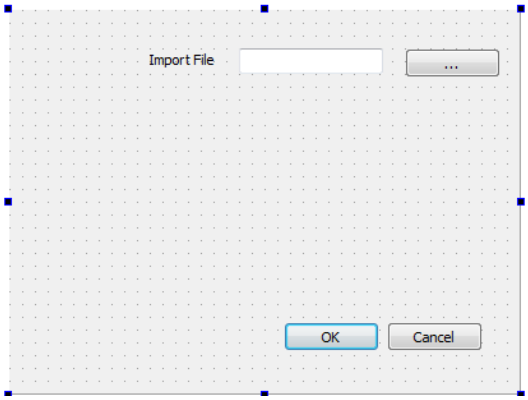


Figure 136 : Interface du futur plugin, sur Qt Designer.

Objet	Classe
SaisieCoordonneesDialogBase	QDialog
button_box	QDialogButtonBox
label	QLabel
lineEdit	QLineEdit
pushButton	QPushButton

Figure 137 : Objets et classes à ajouter pour mettre en place l'interface du plugin.

5.6 Notre plugin

Nous souhaitons créer un plugin qui permet l'export de points sélectionnés dans une couche, au format CSV, en choisissant quels attributs de la couche nous souhaitons exporter.

Pour créer notre plugin, nous nous sommes basés sur un plugin existant, celui de MMQGIS. Il est possible de le télécharger dans les extensions de QGis, et ce plugin va se charger dans la barre de menu, comportant des sous-menus qui contiennent différents plugins (Figure 138).

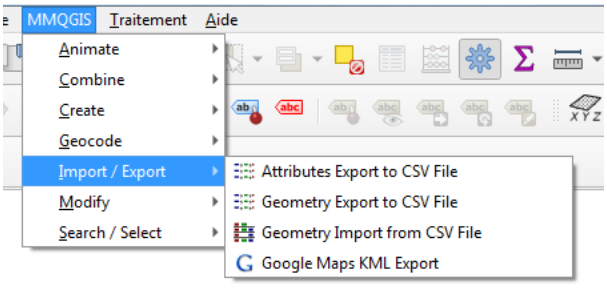


Figure 138 : Différents plugins de MMQGIS, situés dans la barre de menu.

Le plugin qui nous intéresse est le « Attribute Export to CSV File ». Il permet d'exporter les points d'une couche au format CSV, en sélectionnant quels champs l'on souhaite exporter (Figure 139).

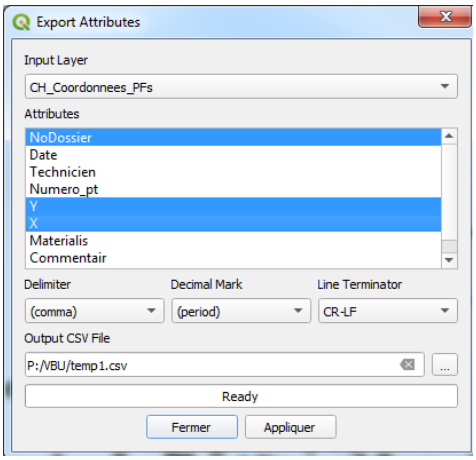


Figure 139 : « Attribute export to CSV file » plugin de MMQGIS.

L'inconvénient de ce plugin est qu'il ne permet pas de choisir quels points exporter ; toute la couche est prise en compte. Nous souhaitons donc ajouter un paramètre supplémentaire au plugin existant, afin de pouvoir exporter en csv juste une sélection de points, et pas l'entièreté de la couche.

En téléchargeant le plugin MMQGIS, nous avons accès au dossier complet (enregistré dans le disque C, dans le même chemin qu'auparavant) contenant les fichiers Python et Qt ui nécessaires à la création du plugin. Il est intéressant de remarquer que dans son dossier de plugin, MMQGIS a créé un library.py (qui correspond à notre plugin.py), un menu.py et un dossier nommé « forms » contenant les fichiers .ui de tous ses plugins (sur Qt Designer). Le menu.py lui permet de créer le menu et les sous menus où seront stockés les différents plugins. Dans notre cas de figure, cette fonctionnalité n'est pas nécessaire.

Fichiers Python

Concernant la partie de code, il faut ouvrir les fichiers Python avec Notepad++. Le plugin que nous avons conçu sur QGis se nomme « plugin_vide », et nous y ferons parfois allusion dans le code.

Nous allons présenter quelques fonctions que nous avons intégrées dans le code, ainsi que des bouts de code représentatifs. Cependant, le code complet des différents fichiers Python se trouve dans l'annexe A2.

Puisque nous nous sommes basés sur un plugin déjà existant, il faut garder en mémoire que nous sommes tributaires du code de MMQGIS, et que par conséquent, il faudra garder certaines de ses

fonctionnalités pour faire fonctionner notre plugin. Les fichiers Python que nous allons modifier et/ou adapter sont les suivants :

init.py
dialog.py
library.py
menu.py

init.py

Nous avons utilisé différentes fonctions (appelées méthodes) dans nos codes. Notamment la fonction *def* de la méthode classFactory, dans le fichier _init_.py. La méthode Factory est un modèle de conception créative, qui construit des instances d'autres classes, où les classes qu'elle crée partagent une classe de base ou une interface commune. Cette méthode sert à simplifier le code de l'application ; au lieu d'utiliser une structure conditionnelle complexe avec des *if*, *elif* et *else* pour déterminer l'implémentation concrète, l'application confie cette décision à un composant séparé qui crée l'objet concret (https://realpython.com/factory-method-python/).

```
def classFactory(iface):

    from .plugin_vide_menu import mmqgis_menu
    return mmqgis_menu(iface)
```

Dialog.py

Dans le fichier dialog.py, nous importons tous les modules qui nous seront utiles pour la réalisation de notre code, grâce à l'instruction *import*.

```
import sys
import csv
import math
import os.path
import operator
```

Lorsque l'on rajoute *from* avant *import*, il est possible d'importer la totalité du module, par l'ajout d'un astérisque.

```
from qgis.core import *
from PyQt5.QtWidgets import *
```

Ici, on charge le module PyQt5.QtWidgets. PyQt est une bibliothèque qui permet d'utiliser le cadre d'interface graphique Qt de Python, et le numéro 5 fait référence à la version la plus récente de Qt.

Par ailleurs, le code par défaut du plugin contient un chemin absolu pour charger le fichier Qt ui (os.path.dirname). Nous l'avons isolé en mode texte (#) et avons décidé de noter le chemin réel, par le biais de la variable sys.path.append.

```
# This loads your .ui file so that PyQt can populate your plugin with the elements from Qt
Designer
#FORM_CLASS, _ = uic.loadUiType(os.path.join(
    #os.path.dirname(__file__), 'plugin_vide_dialog_base.ui'))

import sys
sys.path.append("C:\\Users\\vbuffon\\AppData\\Roaming\\QGIS\\QGIS3\\profiles\\default\\python\
\plugins\\plugin_vide\\forms\\")
```

La variable sys.path est une liste de chaînes de caractères qui détermine le chemin de recherche des modules de l'interpréteur. Elle est initialisée à un chemin par défaut pris dans la variable d'environnement PYTHONPATH, ou à un chemin par défaut intégré si PYTHONPATH n'est pas défini (https://docs.python.org/3/tutorial/modules.html).

De plus, ce fichier contient une classe « mmqgis_dialog » qui englobe les différentes fonctions *def* utilitaires pour la réalisation du plugin, notamment :

```
class mmqgis_dialog(QtWidgets.QDialog):
    def mmqgis_fill_combo_box_with_vector_layers ()
    def mmqgis_find_layer ()
    def mmqgis_initialize_tabular_output_file_widget ()
    def mmqgis_set_status_bar ()
    def mmqgis_status_callback ()
    def mmqgis_temp_file_name ()
```

Ces fonctions seront ensuite appelées dans la deuxième classe « mmqgis_attribute_export_dialog », où il sera question d'importer le formulaire (fichier ui sur Qt Designer), en précisant que l'on importe tout dans le formulaire (import *).

```
from mmqgis_attribute_export_form import *
class mmqgis_attribute_export_dialog(mmqgis_dialog, Ui_mmqgis_attribute_export_form):
```

Library.py

Ce fichier contient différents modules, ainsi que des fonctions permettant de faire fonctionner les boutons de l'interface du plugin. C'est dans ce fichier que l'on va pouvoir définir l'export des entités sélectionnées (et non la couche entière). A deux reprises dans le code, il faudra modifier la fonction layer.getFeatures () par layer.selectedFeatures ().

Menu.py

Le fichier menu.py contient les fonctions pour les sous menus. Nous n'en avons qu'un, celui d'export de point. Nous avons mis notre plugin dans la barre d'outils personnalisée, afin qu'il soit plus accessible et avec les autres boutons.

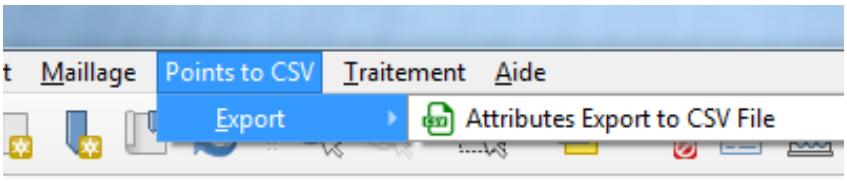


Figure 140 : Notre plugin dans la barre de menu.

Résultats

Le plugin est fonctionnel, il permet l'export de points d'une couche (soit tous les points de la couche, soit uniquement les points sélectionnés).

Il faut sélectionner une couche dans l'arborescence, puis sélectionner les points désirés grâce à l'outil  . Ensuite, cliquer sur le bouton  .

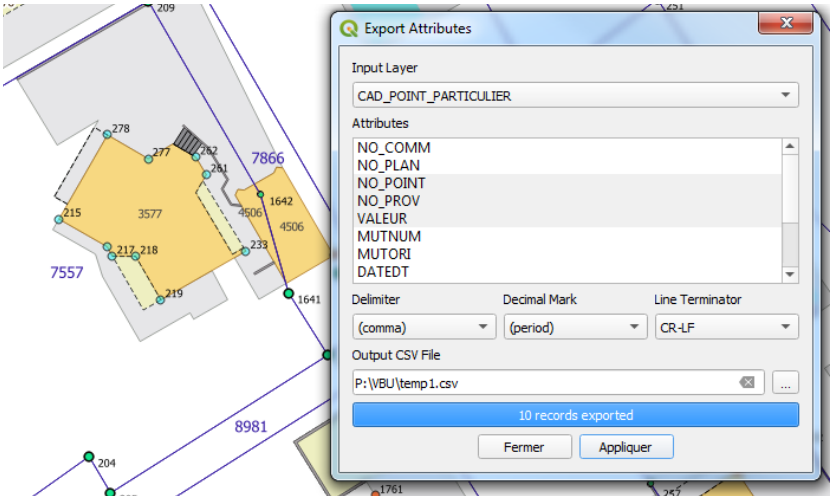


Figure 141 : Export d'attributs en CSV via le plugin créé sur QGis.

L'unique souci est que les valeurs dans l'export de points sont entre guillemets, ce que nous souhaitons éviter. Le théodolite ne reconnait pas ces caractères, il sera donc problématique de lire ce fichier de points sur l'appareil.

NO_POINT,NO_PROV,VALEUR		
217,"0","4"		
262,"0","4"		
215,"0","4"		
278,"0","4"		
233,"0","4"		
216,"0","4"		
277,"0","4"		
219,"0","3"		
261,"0","4"		
218,"0","4"		

Figure 142 : Résultat de l'export d'attributs au format CSV, ouvert sur Excel.

Nous avons tenté de rajouter un bout de code trouvé sur le forum stackoverflow.com pour supprimer les guillemets, sans succès (https://stackoverflow.com/questions/51966824/how-to-remove-double-quotes-in-csv-file-with-python).

```
import csv

    csv.register_dialect('myDialect', delimiter=' ', doublequote=True,
                         quoting=csv.QUOTE_NONE, skipinitialspace='True')

    f = open("normal.csv", 'r')
    f = f.read().replace('"', '').replace('”', '').splitlines()
    normal = csv.reader(f, dialect='myDialect')

    for data in normal:
        print(data, len(data))
```

Finalement, le projet de création de ce plugin n'a pas abouti, nous en sommes restés à une phase intermédiaire. Le plugin est fonctionnel mais il est encore dépendant du plugin d'emprunt (MMQGIS), et il n'exporte pas les points comme souhaité.

Conclusion

Ce stage au sein du bureau Haller Wasser + partner SA a été très instructif et enrichissant. J'ai pu mettre en pratique la plupart des notions que j'avais acquises durant la formation suivie à l'Université de Genève, et j'ai assimilé de nombreuses connaissances et capacités supplémentaires en géomatique. Le fait de développer un outil SIG complet pour le personnel du bureau a été un challenge captivant et très valorisant, et l'utilisation de plusieurs logiciels en simultané s'est révélé très intéressante et avantageuse dans de nombreuses circonstances. En effet, j'ai eu l'opportunité de jongler avec QGis, ArcMap, AutoCAD et FME, ce qui a permis l'obtention de résultats précis et adéquats.

Néanmoins, certains aspects sont restés en suspend et n'ont pu être aboutis, notamment le projet du plugin pour exporter des points au format csv, sur QGis. Pour mener à bien ce projet, il aurait fallu avoir de meilleures connaissances en programmation, et plus de temps pour mieux comprendre comment procéder. J'ai suivi quelques cours en ligne, notamment sur Open Class Rooms ou sur le site d'Anita Graser pour me familiariser avec le langage Python, mais c'est un apprentissage complexe qui demande du temps et de la pratique.

Par ailleurs, certains développements sur FME n'ont pas abouti à des résultats concluants. Plusieurs essais ont dû être effectués avant d'obtenir un développement fonctionnel, concis et compréhensible pour des personnes voyant le développement pour la première fois. D'autre part, certains des développements réalisés ne suivent pas le cheminement le plus direct. Pour ce faire, il aurait fallu mieux étudier les divers *transformers* à disposition, pour employer les plus adéquats.

Concernant la formation en géomatique à l'Université de Genève, celle-ci m'a permis d'obtenir des bases, sur lesquelles j'ai pu m'appuyer tout le long de mon stage. Le fait d'avoir déjà expérimenté la plupart des logiciels et des langages de programmation m'a été d'une grande aide, et j'ai pu puiser des informations dans les cours et/ou rapports rendus. J'estime avoir eu les notions nécessaires pour m'engager dans ce stage, même si plus de pratique au préalable sur QGis aurait été la bienvenue. En effet, nous avons essentiellement travaillé sur ArcGIS Pro et ArcMap durant cette formation, qui sont des logiciels très similaires à QGis, mais pas toutes les entreprises peuvent se permettre la licence plutôt onéreuse. D'autre part, il me semble nécessaire de pouvoir combiner ces logiciels, certains étant parfois plus intuitifs que d'autres, pour avoir toutes les cartes en main. Finalement, les seules connaissances qui m'ont réellement manqué sont celles de programmation Python et de FME. J'ai eu le sentiment que FME est un programme très intuitif et très utile dans de nombreuses situations, et il aurait été intéressant de pouvoir l'expérimenter lors de la formation, car il peut être facilement utilisé en complément aux SIG pour le traitement de données, ou même les remplacer parfois.

Somme toute, les projets réalisés ont enrichi mes connaissances et ma curiosité sur la géomatique en général et sur le métier de géomètre. J'ai eu le l'opportunité d'apprendre de nouvelles notions concernant le cadastre et l'aménagement, et j'ai été formée sur divers aspects de cette profession, notamment en allant sur le terrain, ce qui a éveillé tout mon intérêt pour ce domaine. Et j'ai également eu le plaisir de rencontrer des personnes hors du commun, qui ont fait de mon stage une expérience agréable et inoubliable.

Remerciements

J'aimerais tout d'abord adresser un grand merci à Gaëtan, pour sa patience, sa disponibilité et son partage de connaissances. Merci à Christian et Frédéric, de m'avoir chaleureusement accueillie dans leur bureau, et d'avoir toujours été disponibles.

Un merci tout particulier à Morad, Thomas et Chris, pour tous les moments que nous avons partagés ensemble, pour leur bonne humeur et leur sincère accueil dans le bureau.

Merci à Max, David, Sylvain et Benjamin, mes acolytes de bureau, toujours présents et prêts à donner un coup de main. Merci à Arnaud pour sa précieuse aide en toutes circonstances.

Merci à Alexandre B., Lionel, Sindy, Daniel Q., Céline, Alexandre E.B., Guillaume, Corinne, Rosario, Christophe, Charles, François, Baptiste, Mélanie, Carine, Valentin, Gabriel, Daniel P., Rémy, Julian, Nadia, Etienne, Gilles, Samantha, Cédric, Frédéric T., Axel, Thomas K., Romain, Nicolas, Thierry et Eliane pour leur gentillesse, leur accueil, leur aide et leurs remarques.

Références

QGis

Documentation et forums

https://docs.qgis.org/3.10/en/docs/user_manual/

https://gis.stackexchange.com

https://georezo.net/forum/

https://stackoverflow.com

Action pour charger les orthophotos

https://www.sigterritoires.fr/index.php/travailler-sur-un-catalogue-dimages-aeriennes-ou-satellitaires-avec-qgis-3/

https://gis.stackexchange.com/questions/329547/qgis-3-move-layer-to-group?noredirect=1&lq=1

https://gis.stackexchange.com/questions/75384/add-layer-to-a-qgis-group-using-python?noredirect=1&lq=1

http://osgeo-org.1560.x6.nabble.com/Actions-HowTo-td4534256.html

https://gis.stackexchange.com/questions/69532/how-to-create-a-qgis-action-which-loads-a-raster

https://georezo.net/forum/viewtopic.php?pid=321243

Action pour charger un style de couche

https://gis.stackexchange.com/questions/248189/load-named-style-within-a-group-on-qgis

https://gis.stackexchange.com/questions/338626/loading-layer-styles-qml-file-is-there-a-shortcut-or-toolbar-icon

https://georezo.net/forum/viewtopic.php?id=99341

Montrer les entités sélectionnées dans la mise en page

https://gis.stackexchange.com/questions/111784/displaying-only-selected-features-on-map-in-qgis

FME

https://community.safe.com/s/documentation

Fanout

https://community.safe.com/s/article/fanout-1

https://community.safe.com/s/article/dataset-fanout-doesnt-work-for-services

http://docs.safe.com/fme/2017.1/html/FME_Desktop_Documentation/FME_Workbench/Workbench/Setting_Dataset_Fanout_Properties.htm

Nodes

Nodes :https://developer.apple.com/documentation/scenekit/scnnode/1407958-insertchildnode?language=objc

Transformers FME

https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Transformers/Home.htm

Caching

https://www.safe.com/blog/2018/05/caching-data-fme-evangelist174/

ArcMap

Symbologies/Maplex

https://community.esri.com/thread/195800-arcmap-vbscript-replace-multiple-values-in-one-label

https://gis.stackexchange.com/questions/282367/replacing-text-in-label-expression-of-arcmap

https://desktop.arcgis.com/fr/arcmap/10.3/map/working-with-text/what-is-maplex-.htm

https://www.youtube.com/watch?v=ay3CTz-pRS8

http://help.arcgis.com/en/arcgisdesktop/10.0/pdf/maplex-tutorial.pdf

Python

QGis 3 (create plugin)

https://gis-ops.com/qgis-3-plugin-tutorial-plugin-development-explained-part-2/

https://gis-ops.com/qgis-3-plugin-tutorial-plugin-development-reference-guide/

http://www.qgistutorials.com/en/

http://www.qgistutorials.com/en/docs/3/building_a_python_plugin.html

http://www.qgistutorials.com/fr/docs/building_a_python_plugin.html

https://www.qgistutorials.com/en/docs/3/processing_python_plugin.html

https://www.qgistutorials.com/en/docs/3/getting_started_with_pyqgis.html

https://automating-gis-processes.github.io/site/master/lessons/L7/pyqgis.html

Compilation des ressources

https://automating-gis-processes.github.io/site/master/lessons/L7/pyqgis.html

Plugin MMQGis

http://michaelminn.com/linux/mmqgis/

Python Programming

https://anitagraser.com/pyqgis-101-introduction-to-qgis-python-programming-for-non-programmers/

https://docs.qgis.org/3.10/en/docs/pyqgis_developer_cookbook/plugins/plugins.html#writing-a-plugin

https://www.learnpython.org/

https://openclassrooms.com/fr/courses/4302126-decouvrez-la-programmation-orientee-objet-avec-python/4303896-decouvrez-la-programmation-orientee-objet

https://opensourceoptions.com/tutorials/

https://support.esri.com/fr/technical-article/000018755

https://docs.python.org/3/tutorial/inputoutput.html

https://gis.stackexchange.com/questions/91590/adding-attribute-list-to-combobox-in-qgis-plugin?rq=1

https://stackoverflow.com/questions/51966824/how-to-remove-double-quotes-in-csv-file-with-python

https://realpython.com/factory-method-python/

https://docs.python.org/3/tutorial/modules.html (sys.path.append)

http://www.cc.kyoto-su.ac.jp/~atsushi/Programs/VisualWorks/CSV2HTML/CSV2HTML_PyDoc/ntpath.html (ntpath)

Model builder

https://www.youtube.com/watch?v=eZb5VLTc9-o

https://www.youtube.com/watch?v=Ih6MkK58mog

Mensuration et informations géomètres

https://ge.ch/sitg

https://www.ge.ch/professionnels-mensuration-autres/circulaires-mensuration-officielle

https://www.ge.ch/document/circulaires-dit-2004/annexe/8

https://www.colombes.fr/urbanisme/canevas-topographique-1011.html

https://www.vd.ch/themes/territoire-et-construction/informations-sur-le-territoire/mensuration-officielle/points-fixes/

http://biblio.univ-antananarivo.mg/pdfs/randrianarisonTojonirinaA_ESPA_MAST2_17.pdf)

Autres

http://www.altoa.org/fr/produits-mns-et-mnt.html

https://www.ne.ch/autorites/DDTE/SGRF/SITN/Documents/Brochures/3dlaser_2001.pdf

https://www.e-education.psu.edu/geogvr/node/847

Annexes

A1 : Code pour création d'étiquettes personnalisées (ArcMap)

CAD_NATURE_SOL

```
Function abrev ([GENRE])
 if ( [GENRE]  = "indéfini") then
 abrev = "ind"
 elseif ( [GENRE]  = "inculte") then
 abrev = "inc"
 elseif ( [GENRE]  = "forêt dense") then
 abrev = "fod"
 elseif ( [GENRE]  = "autre boisée") then
 abrev = "fod"
 elseif ( [GENRE]  = "forêt dense") then
 abrev = "ab"
 elseif ( [GENRE]  = "eau stagnante") then
 abrev = "es"
 elseif ( [GENRE]  = "cours d'eau") then
 abrev = "ce"
 elseif ( [GENRE]  = "roselière") then
 abrev = "ros"
 elseif ( [GENRE]  = "route chemin") then
 abrev = "rc"
 elseif ( [GENRE]  = "trottoir") then
 abrev = "tr"
 elseif ( [GENRE]  = "îlot") then
 abrev = "i"
 elseif ( [GENRE]  = "chemin de fer") then
 abrev = "chfer"
 elseif ( [GENRE]  = "place aviation") then
 abrev = "pav"
 elseif ( [GENRE]  = "autre revêtement dur") then
 abrev = "ard"
 elseif ( [GENRE]  = "rocher") then
 abrev = "roc"
 elseif ( [GENRE]  = "éboulis sable") then
 abrev = "esa"
 elseif ( [GENRE]  = "gravière décharge") then
 abrev = "gvd"
 elseif ( [GENRE]  = "autre sans végétation") then
 abrev = "asv"
 elseif ( [GENRE]  = "champ pré") then
 abrev = "chp"
 elseif ( [GENRE]  = "culture intensive") then
 abrev = "cin"
 elseif ( [GENRE]  = "jardin") then
 abrev = "jd"
 elseif ( [GENRE]  = "tourbière") then
 abrev = "trb"
 elseif ( [GENRE]  = "vigne") then
 abrev = "vg"
 elseif ( [GENRE]  = "autre verte") then
```

```
 abrev = "ave"
 end if
End Function

Function FindLabel ( [GENRE], [NIVEAU]  )
 FindLabel = abrev ([GENRE]) + " [Niv= " + [NIVEAU] + "]"
End Function
```

CAD_POINT_LIMITE

```
Function abrev_signe ( [SIGNE]  )
 if ( [SIGNE] = "borne") then
 abrev_signe = "b"
 elseif ( [SIGNE] = "cheville") then
 abrev_signe = "ch"
 elseif ( [SIGNE] = "croix") then
 abrev_signe = "cx"
 elseif ( [SIGNE] = "non matérialisé") then
 abrev_signe = "nmat"
 elseif ( [SIGNE] = "pieu") then
 abrev_signe = "p"
 end if
End Function

Function FindLabel ( [NOPOINT] , [SIGNE] )
 FindLabel = [NOPOINT]  + " (" + abrev_signe ( [SIGNE]  ) + ")"
End Function
```

A2 : Code Python pour le plugin (QGis)

init.py

```
# noinspection PyPep8Naming
def classFactory(iface):  # pylint: disable=invalid-name
    """Load PluginVide class from file PluginVide.

    :param iface: A QGIS interface instance.
    :type iface: QgsInterface
    """
    #
    from .plugin_vide_menu import mmqgis_menu
    return mmqgis_menu(iface)
```

dialog.py

```
import os

from qgis.PyQt import uic
from qgis.PyQt import QtWidgets

# This loads your .ui file so that PyQt can populate your plugin with the elements from Qt
Designer
#FORM_CLASS, _ = uic.loadUiType(os.path.join(
    #os.path.dirname(__file__), 'plugin_vide_dialog_base.ui'))

import sys
sys.path.append("C:\\Users\\vbuffon\\AppData\\Roaming\\QGIS\\QGIS3\\profiles\\default\\python\
\plugins\\plugin_vide\\forms\\")

import csv
```

```
import math
import os.path
import operator

from qgis.core import *

from PyQt5 import QtCore, QtGui, QtWidgets
#from PyQt5.QtCore import *
#from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

try:
    from .plugin_vide_library import *
except:
    from plugin_vide_library import *

# ---------------------------------------------------------------------------
#    mmqgis_dialog - base class for MMQGIS dialogs containing utility functions
# ---------------------------------------------------------------------------

class mmqgis_dialog(QtWidgets.QDialog):
    def __init__(self, iface):
        QtWidgets.QDialog.__init__(self)
        self.iface = iface

    def mmqgis_fill_combo_box_with_vector_layers(self, combo_box):
        # Add layers not in the combo box
        for layer in self.iface.mapCanvas().layers():
            if layer.type() == QgsMapLayer.VectorLayer:
                if combo_box.findText(layer.name()) < 0:
                    combo_box.addItem(layer.name())
                    if layer in self.iface.layerTreeView().selectedLayers():
                        combo_box.setCurrentIndex(combo_box.count() - 1)

        # Remove layers no longer on the map
        removed = []
        for index in range(combo_box.count()):
            found = False
            for layer in self.iface.mapCanvas().layers():
                if layer.name() == combo_box.itemText(index):
                    found = True
                    break
            if not found:
                removed.append(index)

        removed.reverse()
        for index in removed:
            combo_box.removeItem(index)

    def mmqgis_find_layer(self, layer_name):
        if not layer_name:
            return None

        layers = QgsProject.instance().mapLayersByName(layer_name)
        if (len(layers) >= 1):
            return layers[0]

        return None

    def mmqgis_initialize_tabular_output_file_widget(self, file_widget, suffix = ".csv"):
        initial_file_name = self.mmqgis_temp_file_name(suffix)
        file_widget.setFilePath(initial_file_name)
        file_widget.setStorageMode(gui.QgsFileWidget.SaveFile)
        file_widget.setFilter("CSV (*.csv);;Text (*.txt);;All Files (*.*)")

    def mmqgis_set_status_bar(self, status_bar):
        status_bar.setMinimum(0)
        status_bar.setMaximum(100)
        status_bar.setValue(0)
        status_bar.setFormat("Ready")
        self.status_bar = status_bar

    def mmqgis_status_callback(self, percent_complete, message):
        try:
```

```
            if not message:
                message = str(int(percent_complete)) + "%"

            self.status_bar.setFormat(message)

            if percent_complete < 0:
                self.status_bar.setValue(0)
            elif percent_complete > 100:
                self.status_bar.setValue(100)
            else:
                self.status_bar.setValue(percent_complete)

            self.iface.statusBarIface().showMessage(message)

            # print("status_callback(" + message + ")")
        except:
            print(message)

        # add handling of "Close" button
        return 0

    def mmqgis_temp_file_name(self, suffix):
        project = QgsProject.instance()

        home_path = project.homePath()
        if not home_path:
            home_path = os.getcwd()

        for x in range(1, 10):
            name = home_path + "/temp" + str(x) + suffix
            if not os.path.isfile(name):
                return name

        return home_path + "/temp" + suffix

from mmqgis_attribute_export_form import *

class mmqgis_attribute_export_dialog(mmqgis_dialog, Ui_mmqgis_attribute_export_form):
    def __init__(self, iface):
        mmqgis_dialog.__init__(self, iface)
        self.setupUi(self)
        self.mmqgis_set_status_bar(self.status)
        self.buttonBox.button(QtWidgets.QDialogButtonBox.Apply).clicked.connect(self.run)

        self.mmqgis_fill_combo_box_with_vector_layers(self.input_layer_name)

        self.field_delimiter.addItems(["(comma)", "(semicolon)", "(space)"])
        self.line_terminator.addItems(["CR-LF", "LF"])
        self.decimal_mark.addItems(["(period)", "(comma)"])

        self.input_layer_name.currentIndexChanged.connect(self.set_attributes)
        self.set_attributes()

        self.mmqgis_initialize_tabular_output_file_widget(self.output_csv_name)

    def refresh_layers(self):
        self.mmqgis_fill_combo_box_with_vector_layers(self.input_layer_name)

    def set_attributes(self):
        self.attributes.clear()
        layer = self.mmqgis_find_layer(self.input_layer_name.currentText())
        if (layer == None):
            return

        for field in layer.fields():
            self.attributes.addItem(field.name())

        # Decimal mark is dependent on locale (comma instead of decimal point in Europe)
        # This doesn't use local override in settings (although it should)
        # Idiot check to make sure it's either a period or comma

        # QGIS configured country codes (locale/userLocale) are not
        # usable with python locale, so just use QLocale from user interface

        if (self.locale().decimalPoint() == '.'):
            self.decimal_mark.setCurrentIndex(self.decimal_mark.findText("(period)"))
```

```
            self.field_delimiter.setCurrentIndex(self.field_delimiter.findText("(comma)"))

        else:
            self.decimal_mark.setCurrentIndex(self.decimal_mark.findText("(comma)"))
            self.field_delimiter.setCurrentIndex(self.field_delimiter.findText("(semicolon)"))

    def run(self):
        input_layer = self.mmqgis_find_layer(self.input_layer_name.currentText())

        attribute_names = []
        for x in self.attributes.selectedItems():
            attribute_names.append(str(x.text()))

        if str(self.field_delimiter.currentText()) == "(space)":
            field_delimiter = " "
        elif str(self.field_delimiter.currentText()) == "(semicolon)":
            field_delimiter = ";"
        else:
            field_delimiter = ","

        if str(self.line_terminator.currentText()) == "LF":
            line_terminator = "\n"
        else:
            line_terminator = "\r\n"

        if str(self.decimal_mark.currentText()) == "(comma)":
            decimal_mark = ','
        else:
            decimal_mark = '.'

        output_csv_name = str(self.output_csv_name.filePath())

        message = mmqgis_attribute_export(input_layer, attribute_names, output_csv_name,
            field_delimiter, line_terminator, decimal_mark, self.mmqgis_status_callback)

        if message != None:
            QMessageBox.critical(self.iface.mainWindow(), "Attribute Export", message)
```

Library.py

```
import io
import re
import csv
import sys
import ssl
import time
import json
import math
import locale
import random
import os.path
import operator
import tempfile
import urllib.parse
import urllib.request
import xml.etree.ElementTree

from qgis.core import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *

# Used instead of "import math" so math functions can be used without "math." prefix
from math import *


def mmqgis_attribute_export(input_layer, attribute_names, output_csv_name, \
        field_delimiter = ",", line_terminator = "\n", decimal_mark = ".", \
        status_callback = None):

    # Error checks
```

```
    if (not input_layer) or (input_layer.type() != QgsMapLayer.VectorLayer) or
(input_layer.featureCount() <= 0):
        return "Invalid layer"

    if not attribute_names:
        return "No attributes specified for export"
    layer_options = []
    if line_terminator == "\r\n":
        layer_options.append("LINEFORMAT=CRLF")
    else:
        layer_options.append("LINEFORMAT=LF")

    if field_delimiter == ";":
        layer_options.append("SEPARATOR=SEMICOLON")
    elif field_delimiter == "\t":
        layer_options.append("SEPARATOR=TAB")
    elif field_delimiter == " ":
        layer_options.append("SEPARATOR=SPACE")
    else:
        layer_options.append("SEPARATOR=COMMA")

    if not decimal_mark:
        decimal_mark = "."


    # Build field list
    fields = QgsFields()
    attribute_indices = []

    for attribute in attribute_names:
        found = False
        for index, field in enumerate(input_layer.fields()):
            if field.name() == attribute:
                fields.append(field)
                attribute_indices.append(index)
                found = True
                break
        if not found:
            return "Invalid attribute name: " + attribute

    if not attribute_indices:
        return "No valid attributes specified"

    # Create file writer
    outfile = QgsVectorFileWriter(output_csv_name, "utf-8", fields, \
        QgsWkbTypes.Unknown, driverName = "CSV", layerOptions = layer_options)

    if (outfile.hasError() != QgsVectorFileWriter.NoError):
        return "Failure creating output file: " + str(outfile.errorMessage())


    # Iterate through each feature in the source layer
    for index, feature in enumerate(input_layer.selectedFeatures()):
        if ((index % 50) == 0) and status_callback:
            if status_callback(100 * index / input_layer.featureCount(), \
                    "Exporting " + str(index) + " of " + str(input_layer.featureCount())):
                return "Export attributes cancelled on feature " + str(index)

        attributes = []
        for x in attribute_indices:
            attributes.append(feature.attributes()[x])

        newfeature = QgsFeature()
        newfeature.setAttributes(attributes)
        outfile.addFeature(newfeature)
```

```
    del outfile

    if status_callback:
        status_callback(100, str(len(input_layer.selectedFeatures())) + " records exported")

    return None
```

NB : le texte en rouge correspond à ce que nous avons modifié par rapport au code de MMQGIS.

menu.py

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from qgis.core import *

from .plugin_vide_dialog import *

# ----------------------------------------------

class mmqgis_menu:
    def __init__(self, iface):
        self.iface = iface
        self.mmqgis_menu = None

    def mmqgis_add_submenu(self, submenu):
        if self.mmqgis_menu != None:
            self.mmqgis_menu.addMenu(submenu)
        else:
            self.iface.addPluginToMenu("&mmqgis", submenu.menuAction())

    def initGui(self):
        # Uncomment the following two lines to have MMQGIS accessible from a top-level menu
        self.mmqgis_menu = QtWidgets.QMenu(QCoreApplication.translate("mmqgis", "Points to
CSV"))

self.iface.mainWindow().menuBar().insertMenu(self.iface.firstRightStandardMenu().menuAction(),
self.mmqgis_menu)


        # Import / Export Submenu
        self.import_export_menu = QtWidgets.QMenu(QCoreApplication.translate("mmqgis",
"&Export"))
        self.mmqgis_add_submenu(self.import_export_menu)

        icon =
QIcon("C:\\Users\\vbuffon\\AppData\\Roaming\\QGIS\\QGIS3\\profiles\\default\\python\\plugins\\
plugin_vide\\icons\\mmqgis_attribute_exports.png")
        self.attribute_export_action = QtWidgets.QAction(icon, "Attributes Export to CSV
File", self.iface.mainWindow())
        self.attribute_export_action.triggered.connect(self.attribute_export)
        self.import_export_menu.addAction(self.attribute_export_action)


    def unload(self):
        if self.mmqgis_menu != None:
            self.iface.mainWindow().menuBar().removeAction(self.mmqgis_menu.menuAction())
        else:
            self.iface.removePluginMenu("&mmqgis", self.import_export_menu.menuAction())

        # This one button in the plugins toolbar is for the South Derbyshire District Council
(7/14/2013)
        # self.iface.removeToolBarIcon(self.search_action)


    def attribute_export(self):
```

```
try:
    self.attribute_export_dialog.refresh_layers()
except:
    self.attribute_export_dialog = mmqgis_attribute_export_dialog(self.iface)

self.attribute_export_dialog.exec_()
```