LAMIA – Documentation technique

# Introduction

Ce document a pour but de donner les éléments de contexte du développement de LAMIA, de détailler le fonctionnement du code et de fournir suffisamment d’informations techniques pour permettre à toute personne le souhaitant de rapidement commencer à développer de nouvelles fonctionnalités.

# Démarche générale, LICENCE

Nous cherchons à aider à la construction de base de données robustes et cohérentes qui pourront vivre pendant des années et permettre le développement d’une connaissance forte des infrastructures afin de permettre le passage d’une approche de déconstruction/reconstruction à une approche de gestion patrimoniale et de maintenance préventive.

Notre but final est de permettre une réduction des coûts de maintenance des grandes infrastructures linéaires telles que les digues, les routes ou encore les ponts, tout en en limitant drastiquement l’impact écologique.

Nous cherchons à fournir à tous les secteurs public et privé un outil simple d’utilisation pour la collecte, la maintenance et la vie d’une donnée de grande qualité sur les infrastructures linéaires.

Le projet LAMIA est développé et maintenu par ARTELIA et diffusé sous licence GNU GPL v3 (ou postérieure).

# Ressources nécessaires au lancement

LAMIA est un module complémentaire pour Qgis 2.18 et supérieures se manifestant sous la forme d’un panneau supplémentaire dans l’application.

Pour installer LAMIA, les bibliothèques suivantes sont nécessaires :

* pyqgis
* Numpy
* Matplotlib
* Networkx
* Xlrd (excel reader)
* Pillow

Si vous disposez de toutes ces bibliothèques et que le python que Qgis utilise y a bien accès, il vous suffit d’ajouter LAMIA au dossier des extensions Qgis pour pouvoir commencer à l’utiliser.

Vous pouvez aussi télécharger LAMIA directement depuis le dépôt des extensions officielles Qgis.

# FONCTIONNEMENT GENERAL

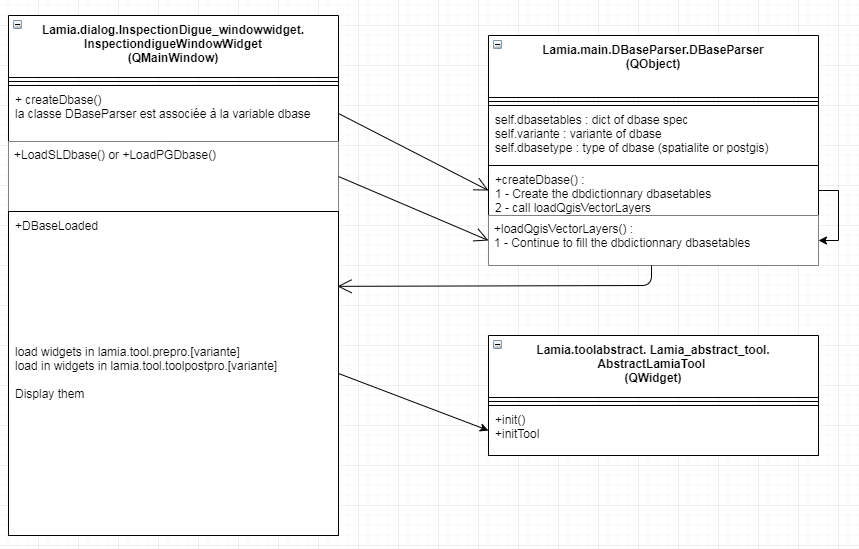
# Structure du code

## Structuration générale

Le code repose principalement sur trois classes.

* La classe « Lamia.dialog.InspectionDigue\_windowwidget.InspectiondigueWindowWidget », qui est chargée au démarrage. Il s’agit du QDockWidget apparaissant au click sur l’icone Lamia dans l’interface QGis.
* La classe « Lamia.main.DBaseParser.DBaseParser ». Il s’agit d’un QObject chargé de l’ensemble des intéractions avec la base de données.
* La classe « Lamia.toolabstract. Lamia\_abstract\_tool. AbstractLamiaTool » qui est un widget abstrait dont héritent tous les widgets apparaissant sur l’écran « propriétés ».

L’activité lors de l’ouverture de la base de données est schématisée ci-dessous :



InspectiondigueWindowWidget  est créé lors de l’ouverture de Lamia. Au chargement/création de la base de données, la classe DBaseParser est créée en variable d’InspectiondigueWindowWidget et cette classe se charge de la création et de la lecture de la base de données.

Ensuite, InspectiondigueWindowWidget.DBaseLoaded  charge l’ensemble des widgets (tous héritant de AbstractLamiaTool) apparaissant dans l’écran en bas à droite du InspectiondigueWindowWidget.

Ainsi, le code est structuré de la façon suivante :

|  |  |
| --- | --- |
| **Dossier** | **Contenu** |
| +---dialog | Contient les fichiers sources des interfaces graphiques des boites de dialogues de LAMIA ainsi que les codes python pour les gérer.  Il contient notamment InspectiondigueWindowWidget . |
| +---main | Contient le main de LAMIA qui importe le DBaseParser et le DBaseParser qui initialise l'environnement LAMIA. |
| +---toolabstract | Contient Lamia\_abstract\_tool.py qui est la classe abstraite dont héritent tous les widgets présents en bas à droite de InspectiondigueWindowWidget . |

Les éléments propres à la création et à la manipulation de la base de données sont ici :

|  |  |
| --- | --- |
| **Dossier** | **Contenu** |
| +---DBASE | Dossier contenant tous les éléments nécessaires à la création des bases de données |
| | +---BPU | Dossier contenant les bordereaux de prix permettant le chiffrage automatique des travaux |
| | +---create | Dossier contenant tous les fichiers excel embarquant la description des tables ainsi que les nomenclatures |
| | +---sqlite\_base | Dossier contenant la base SQLITE utilisée pour amorcée la création des bases de données |
| | +---style | Contient les feuilles de styles utilisées pour l'affichage de l'interface LAMIA |
| | \---utils | Contient des ressources d'affichage |

Tous les widgets apparaissant en bas à droite d’InspectiondigueWindowWidget et héritant de AbstractLamiaTool sont stockés ici :

|  |  |
| --- | --- |
| **Dossier** | **Contenu** |
| +---toolpostpro | Outils de post-production (synthèses, exports, rapports, …) |
| | +---Base2 | Outils génériques d'import, d'export, de synthèse, de gestion des paths, de génération des rapports, d'étude de coûts, … |
| | +---[Type métier] | Outils de post-traitement adaptés au métier. |
| \---toolprepro | Interfaces graphiques et classes pythons permettant de manipuler la base de données |
| | +---Base2 | Outils génériques |
| | +---[Type métier] | Outils d’alimentation de la base de données adaptés au métier. |

En plus de ces packages, les packages suivants sont également utilisés.

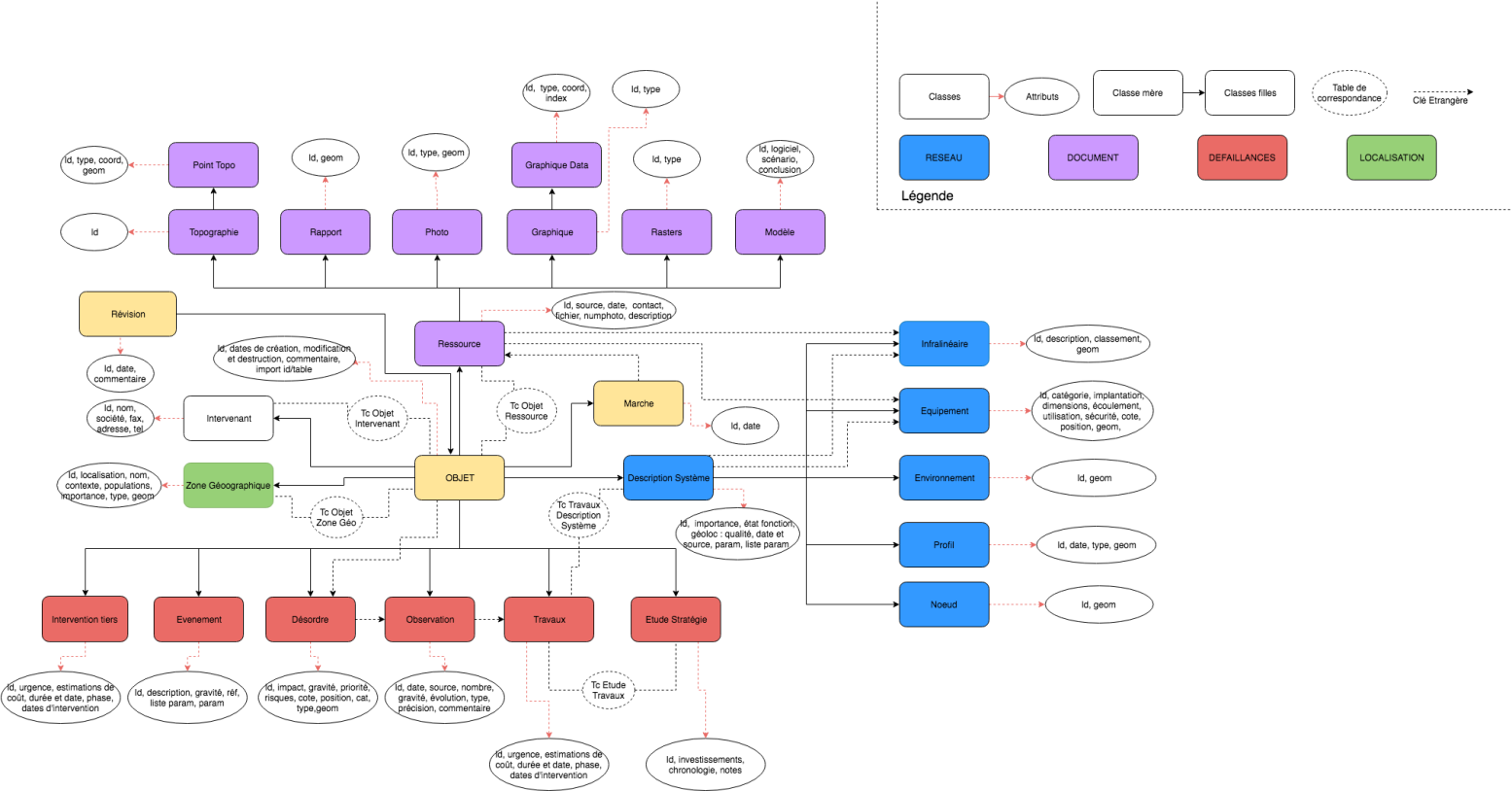
|  |  |
| --- | --- |
| **Dossier** | **Contenu** |
| +---config | Stock les données récemment utilisées à conserver en mémoire comme l'adresse des dernières bases utilisées |
| +---gps | Contient les outils de connexion et d'exploitation des GPS |
| +---html | Contient des outils de mise en page (js, css, images, polices, …) |
| +---icons | Contient les icones utilisés dans LAMIA |
| +---lamiautils | Vide |
| +---libs | Continent certaines librairies embarquées directement dans LAMIA, souvent parce qu'elles ont été modifiées |
| | +---cloudant | Librairie de connexion à une base NoSQL (lien avec SIRS Digues) |
| | +---cloudant\_2\_10 | Librairie de connexion à une base NoSQL (lien avec SIRS Digues) |
| | +---pyqtgraph | Librairie de génération de graphiques |
| | \---xlrd | Librairie de lecture de fichier Excel utilisée pour la création des bases de données |
| +---maptool | Contient les éléments d'interaction avec la cartographie de Qgis |
| +---toolgeneral | Contient des modules complémentaires pour LAMIA |
| | +---LAMIA\_to\_SIRS | Permet de faire la passerelle depuis LAMIA vers SIRS Digues |
| | +---LAMIA\_to\_SIRS\_VCouch1\_7 | Permet de faire la passerelle depuis LAMIA vers SIRS Digues V1.7 |
| | +---SIRS\_to\_LAMIA | Permet de faire la passerelle depuis SIRS Digues vers LAMIA |
| | \---SIRS\_to\_LAMIA\_VCouch1\_7 | Permet de faire la passerelle depuis SIRS Digues vers LAMIA |

# La base de données dans Lamia

## Schéma fonctionnel de la donnée

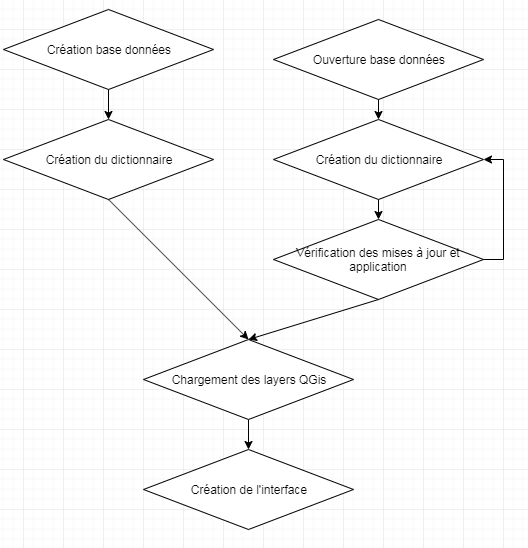
Toutes les bases de données LAMIA sont construites selon le même template, présenté plus bas.

Toutefois, pour adapter les outils aux besoins de chaque thématique, cette base « socle » est régulièrement complétée par de nouveaux champs et une nomenclature spécifique.



## Initialisation de la base de de données dans Lamia

Le processus d‘initialisation de la base de données est schématisé ci-dessous.



Ces étapes (hormis la dernières) sont gérées par la classe DBaseParser.

### Préalable : Définition du type de base de données

La base de données a deux paramètres généraux, qui sont le « type de métier » et la « variante ».

* Le type de métier correspond à des jeux de noms de champs de la base de données propre à un métier (p.ex, les infrastructures linéaires en eau potable ont une caractéristique « diamètre », alors qu’en voirie elles auront une caractéristiques « largeur »).
* La variante correspond à des valeurs possibles dans la base de données différentes selon la variante (p.ex, en eau potable, tel client souhaite dans le champ « visitable » la valeur « obstrué » alors qu’un autre veut « non visible »).

Ces éléments sont définis lors de la création de la base de données.

le type de métier et la variante sont un préalable obligatoire pour la création du dictionnaire.

### Création du dictionnaire DBaseParser.dbasetables

* **Principe**

Les fichiers utilisés pour la création du dictionnaire sont dans le répertoire Lamia\DBASE\create.

Le premier fichier lu est le fichier Base2\_X\_X.xls, qui contient les tables et les champs communs à l’ensemble des bases de données indifféremment du métier et de la variante.

Le deuxième fichier lu est le fichier Base2\_[métier]\_X\_X.xls qui contient les éléments de base de données propres au métier choisi. Ce fichier écrase et complète le premier fichier lu.

Notons que le X\_X correspond à la version de la base de données.

* A la création de la base de données, c’est la dernière version qui est employée.
* Lors de l’ouverture de la base de données, Lamia vérifie si c’est la dernière version. Si non, il créé les champs manquant et exécute les lignes de codes situées dans Base2\_[métier]\_X\_X.txt.
* **Nomenclature utilisée dans le fichier .xls**

Les deux premières lignes commencent par #DJAN et #QGIS et indiquent les vues utilisées par respectivement django et qgis.

la variable #SPATIALINDEX indique la création ou non d’in index spatial.

#VARIANTE indique la présence ou non de variante. S’il y en a au moins deux, il convient de mentionner leur nom (dont la première « Lamia ») dans les colonnes 6+3n.

La partie suivante du tableau Excel contient, sur chaque ligne, les colonnes de la table. Y sont définies :

* + Le nom des champs (Nom\_champ) ;
  + Le type en PostGIS (Type postgis) ;
  + Une liaison de type clef étrangère (Option) au format « Table(champ) » ;
  + La description (Description) ;
  + Le champ dont le champ hérite ses conditions de nomenclature (si certaines options ne s’affichent que dans certains cas de figure) (Parentfield) ;
  + Les valeurs à utiliser dans LAMIA (Affichage) ;
  + La donnée à stocker en fonction de la valeur sélectionnée dans l’interface graphique (Stockage) ;
  + Les valeurs du champ parent sous lesquelles la valeur du champ actuelle doit être affichée (Parentfielddependant) ;
  + Au besoin, les autres nomenclatures à utiliser (sous le nom de la variante).

Ce fichier est parcouru chaque onglet amène la création d’une table dans la base de données, avec les champs adaptés.

* **Dictionnaire créé : DBaseParser.dbasetables**

Le dictionnaire créé est ainsi :

{...{tablename : {'order' : order,

'geom' : geometry type ,

'widget' : the table widget,

'djangoviewsql' : sql statement for initial django view creation

'qgisviewsql' : sql statement for initial qgis view creation

'qgisSLviewsql' : sql statement for initial qgis view creation - spatialite compatible

'qgisPGviewsql' : sql statement for initial qgis view creation - postgis compatible

'exportviewsql' : sql statement for initial special view creation

'layer' : real layer

'layerqgis' : view layer with parent fields

'layerdjango' : view layer with parent fields

'showinqgis' : display layer in canvas

'scale' : visibility scale

'spatialindex' : create a spatialite spatial index

'fields' : OrderedDict{...{fieldname : {'PGtype' : PostGis type (integer not null...)

'SLtype' : spatialite type (integer not null...)

'FK' (optional) : foreign key definition

'ParFldCst (optional) : name of the parent field for contraint

'Cst' (optional) : list of constraint : [description,value,[parent field consraint value]}

...}

}

...}

### Création éventuelle de la base de données

La base de données est créé avec des reques SQL à partir de DBaseParser.dbasetables.

### Chargement des layers qgis

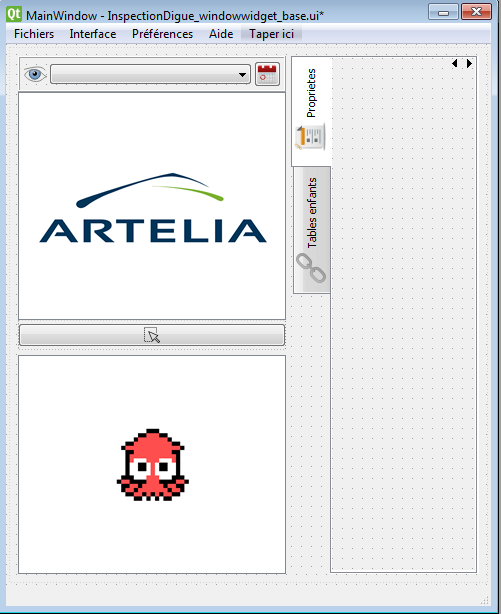
les qgsvectorlayers sont stockés dans tablename.'layerqgis'.

# Variables et constantes du programme

## La Classe InspectiondigueWindowWidget

### Les sous-widget

stackedWidget\_main



MaintabWidget

MaintreeWidget

ElemtreeWidget

### Les variables notables

self.uifields : tableau contenant les widgets de prétraitement pour l’interface terrain

self.uidesktop : tableau contenant les widgets de prétraitement pour l’interface bureau

self.uipostpro : tableau contenant les widgets de prétraitement pour l’interface postpro

self.dbase : une instance du DBaseParser

self.gpsutil : instance de la classe de communication avec le GPS.

### Les principales fonctionnalités

* Interface avec DBaseParser : création, chargement et changement de base
* affichage des messages dans le canvas QGis
* Changement d’interface ;
* Interface avec la GPS ;
* changement de thème

## La Classe DBaseParser

### Les variables notables

self.visualmode : stocke le type d’interface utilisé (0 : terrain ; 1 : bureau ; 4 : posttraitement)

self.dbasetype : le type de base de données (spatialite, postgis)

self.type : le type de métier

self.variante : le type de variante

self.version : la version du fichier Base2.xls

self.workversion : la version du fichier Base2\_[métier].xls

self.dbasetables : le dictionnaire permettant l’accès à toutes les données contenues dans les fichiers.xls

self.crsnumber : le code EPSG de la projection utilisée

### Les principales fonctions

createDbase : créé une base de données

loadQgisVectorLayers : charge une base de données

importDbase : importe une base de données

readDbDictionnary : construit self.dbasetables à partir des variables type et variantes définies.

query : permet de passer une requête SQL

updateQueryTableNow : permet de passer une requête SQL avec une table XXX\_now : filtre les éléments supprimés ou appartenant à une version archivée.

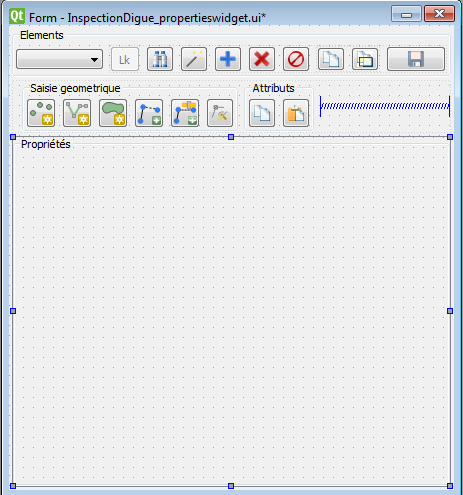
getValuesFromPk(self, dbasename, fields, pk) : obtenir une valeur d’une table pour un champ

isAttributeNull(self ,attr) : permet de savoir usi une valeur est NULL

## La Classe AbstractLamiaTool

### Les sous-widget

frame\_edit



frame\_editing

groupBox\_elements

comboBox\_featurelist

groupBox\_properties

groupBox\_attributes

groupBox\_geom

### Les variables notables - dans le cas d’une utilisation « simple »

self.dbase : l’instance de DBaseParser en cours

self.CAT : le nom e la catégorie dans laquelle apparait le widget dans le MaintreeWidget du InspectiondigueWindowWidget

self.NAME : le nom de la souscatégorie dans laquelle apparait le widget dans le MaintreeWidget du InspectiondigueWindowWidget

self.windowdialog : l’instance de InspectiondigueWindowWidget

### Cas pratique : création d’un nouveau module de post-traitement

Nous voulons créer un module de post-traitement pour le métier « éclairage ».

Dans ce cas, il faut créer un fichier test\_module.py dans le répertoire lamia/toolpostpro/Base2\_eclairage.

Ensuite, il convient de créer un widget avec QtDesigner et de l’enregistrer sous 'test\_module.ui' dans le même répertoire.

Ce fichier aura la structure minimale suivante :

import os

from ...toolabstract.Lamia\_abstract\_tool import AbstractLamiaTool

from qgis.PyQt import uic

from qgis.PyQt.QtWidgets import (QWidget)

class CostTool(AbstractLamiaTool):

TOOLNAME = 'test\_module'

def \_\_init\_\_(self, dbase, dialog=None, linkedtreewidget=None, gpsutil=None,parentwidget=None, parent=None):

super(CostTool, self).\_\_init\_\_(dbase, dialog, linkedtreewidget, gpsutil,parentwidget, parent=parent)

def initTool(self):

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Main spec

self.CAT = 'Synthese'

self.NAME = 'Couts'

self.visualmode = [4]

self.groupBox\_elements.setParent(None)

self.frame\_editing.setParent(None)

def initFieldUI(self):

if self.userwdgfield is None:

self.userwdgfield = UserUI()

class UserUI(QWidget):

def \_\_init\_\_(self, parent=None):

super(UserUI, self).\_\_init\_\_(parent=parent)

# self.setupUi(self)

uipath = os.path.join(os.path.dirname(\_\_file\_\_), 'test\_module.ui')

uic.loadUi(uipath, self)

La classe UserUI permet de créer le widget.

Ensuite, la classe CostTool est créée. il convient :

* de créer la variable TOOLNAME pour que ce fichier soit reconnu comme un fichier de post-process.
* de créer init() de façon à surcharger la classe parente AbstractLamiaTool
* de surcharger la fonction initTool où :
  + CAT et NAME sont les noms qui apparaissent dans le MaintreeWidget du InspectiondigueWindowWidget ;
  + visualmode indique qu’il s’agit de l’afficher quand l’interface post-traitement est choisie ;
  + groupBox\_elements et frame\_editing sont cachées.
* de surcharger initFieldUI pour charger le widget.