Manuel Technique Datüm

# Intervention sur les scans

Pour pouvoir intervenir sur un scan (n’importe lequel des scans présent dans le dossier DATUM/scan) et vérifier que les modifications sont bien prises en compte, il faut réaliser une petite série de manipulations dans l’espace de qualification :

* Copier les dossiers data et json, ainsi que le fichier param\_scan.py dans le dossier scan.
* Enlever la mise en commentaire de la ligne d’import de param\_scan dans le scan.
* Effectuer les modifications souhaitées.
* Lancer le script.
* Enlever la mise en commentaire de la ligne d’import de param\_scan dans FIN.py.
* Lancer FIN.py pour produire les JSON, sinon les nouveaux résultats ne seront pas exportés et ne seront visibles que dans les tables Oracle.
* Déplacer le contenu du dossier DATUM/scan/json dans le dossier DATUM/json (donc dans celui d’origine, dans lequel le tableau de bord vient récupérer les informations).
* Lancer Datüm et observer les changements.

Lorsque les modifications sont satisfaisantes, penser à remettre Datüm dans son état normal :

* Commenter les lignes d’import.
* Supprimer les dossiers et fichiers (dont scan.txt et scan\_debug.txt) ajoutés dans scan.
* Mise en production du script modifié par copié/collé du fichier .py dans l’espace de production.

# Traitement des dates de dernière modification

Pour plus d’informations sur les recherches et travaux menés en ce sens, se référer à recherche\_date\_maj.docx.

## Les dates systèmes d’Oracle

Après plusieurs tests menés du 15 au 17 Mai 2018 pour isoler les éléments sources d’incertitudes, voici ce que nous pouvons affirmer sur ces dates :

* Les modifications de type **DML** (une de ces trois action : INSERT, UPDATE ou DELETE) sont enregistrées dans le **TIMESTAMP** de la table **ALL\_TAB\_MODIFICATIONS**. Elles prennent en compte ajout / suppression / modification de cellules ou de lignes en suivant ces conditions :
  + Altération inférieure à 10% du segment (un segment = table pour Oracle)
  + Si altération supérieure à 10%, la date est réinitialisée et n’apparaît plus dans la table système (mais elle reste enregistrée dans **DATÜM**.
  + Action périodique automatisée sur l’ensemble des bases par la DSIT (qui a lieu au changement de semaine ou de mois).
  + Moissonnage de statistiques Oracle effectué (visiblement à 22h tous les jours).
  + Si autre horaire que 22h, l’heure indiquée est fiable à environ 15min. Toutes les tables ayant subi une DML dans ce laps de temps auront le même TSP.
  + Pas de différenciation du comportement pour tables ou classes d’entités.
* Les modifications de type **DDL** (voir ci-dessous) sont enregistrées dans le **LAST\_DDL\_TIME** de la table **ALL\_OBJECTS**. Elles prennent en compte plusieurs éléments distincts :
  + Ajout / suppression / modification de champs (enregistrement LDT instantané)
  + Ajout / suppression / modification des droits (enregistrement LDT instantané)
  + Annule et remplace de la totalité du contenu de la table avec script **FME** ou commande ‘charger’ d’ArcMap.
  + Pas de différenciation du comportement pour tables ou classes d’entités.

On sait que le **TIMESTAMP** de **ATM**(*ALL\_TAB\_MODIFICATIONS)* est renseigné pour environ 280 classes d’entités et qu’il est susceptible d’augmenter au fur et à mesure, puisqu’elles n’ont commencé à apparaître qu’avec le passage de Brest métropole sur des bases de données **Oracle**.

Il peut arriver, lors de modifications supérieures à 10% du segment, que cette date soit effacée de la table. Pour que **DATÜM** affiche des données cohérentes, il a été décidé de ne plus réinitialiser la colonne *‘tab\_datemodif’* de *‘ZZZ\_TDB\_TABLESDE’* et de simplement écraser les dates à chaque nouveau scan si la valeur de **TIMESTAMP** n’est pas nulle.

Pour compléter les informations récupérées dans **ATM**, nous avons conservé les **LAST\_DDL\_TIME** plus récents qui viennent aussi apporter leur date lors d’absence de **TIMESTAMP**.

## Logigramme récapitulatif

ATM = ALL\_TAB\_MODIFICATIONS et AO = ALL\_OBJECTS (tables systèmes d’Oracle)

TSP = TIMESTAMP (champ d’ATM) et LDT = LAST\_DDL\_TIME (champ d’AO)

CE = Classe d’Entité des bases SDE

= Oui

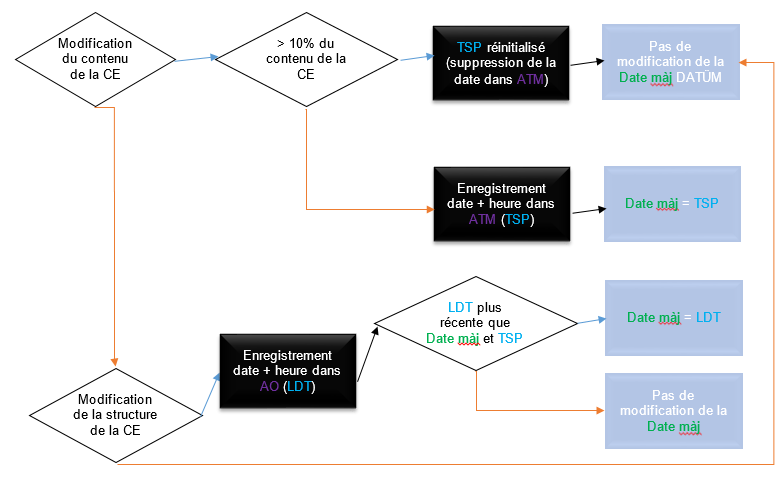
= Non

Oracle

= Action automatique d’Oracle

DATÜM

= Action liée au scan de DATÜM



## Scripts JeuxDonnees\_Maj.py et JeuxDonneesMajSuivi.py

### JeuxDonnees\_Maj.py

Création d’un dictionnaire dont chaque clé est un le nom d’un jeu de données dans ZZZ\_TDB\_TABLESDE (moins les quatre premiers caractères : SIG.) prenant pour valeur le nom complet et la date de modification :

dicoJeux **=** {row.getValue('TAB\_NOM')[4**:**].upper()**:**(row.getValue('TAB\_NOM'),row.getValue('TAB\_DATEMODIF')) **for** row **in** arcpy.SearchCursor(ZZZ\_TDB\_tablesde)}

Récupération des dates systèmes en suivant la méthodologie détaillée précédemment avec des requêtes SQL de type :

sql\_ATM**=**"select TABLE\_NAME, " \  
 "to\_char((TIMESTAMP), 'YYYY-MM-DD HH24:MI:SS') " \  
 "from ALL\_TAB\_MODIFICATIONS " \  
 "where TABLE\_OWNER = 'SIG'"

Ici, to\_char((*date*), ‘*format de date souhaité’*) est nécessaire car depuis la version 10.5 d’ArcGIS, les dates inscrites dans les tables sont dans un format non supporté par JSON. Il est donc converti en string, résolvant le problème d’export d’une date au format JSON.

Pour chaque résultat des requêtes SQL, il recherche ensuite une date si le suivi éditeur est activé. Bien qu’exceptionnellement actif, la date qu’il fournit est totalement fiable :

**if** desc.editorTrackingEnabled**:** rows **=** arcpy.SearchCursor(rowSQL[0])  
 **for** row **in** rows**:  
 if** row.getValue(desc.editedAtFieldName)**!=** None**:  
 if** row.getValue(desc.editedAtFieldName) **>** date\_ATM**:** date\_ATM **=** row.getValue(desc.editedAtFieldName)

Enregistrement des valeurs du dictionnaire des dates de modification dans la table, avec vérification que chaque jeu de données est présent :

dicoRec**=** {dicoJeux[x][0]**:**dicoJeux[x][1] **for** x **in** dicoJeux.keys()} *#dico pour enr. date dans DATUM*rows **=** arcpy.UpdateCursor(ZZZ\_TDB\_tablesde)

**for** row **in** rows**:  
 if** row.getValue('TAB\_NOM') **in** dicoRec.keys()**:** row.setValue('TAB\_DATEMODIF', dicoRec[row.getValue('TAB\_NOM')])  
 rows.updateRow(row)  
**del** rows

### JeuxDonnes\_MajSuivi.py

Création d’un dictionnaire des différentes planifications accessibles dans l’éditeur de métadonnées avec leur correspondance numérique de jours ou non planifiée si pas de conversion possible. Exceptions faites pour figée et continue qui ont des traitements particuliers par la suite :

*#Planification correspondant en jours*dicoPlanif **=** {**u'quotidienne': u'continue'**, **u'hebdomadaire': u'continue'**, **u'mensuelle':** 31, **u'bimensuelle':** 62, **u'trimestrielle':** 93, **u'biannuelle':** 183, **u'annuelle':** 366, **u'tous les 2 ans':** 732, **u'tous les 3 ans':** 1098, **u'tous les 4 ans':** 1464, **u'tous les 5 ans':** 1830, **u'tous les 10 ans':** 3660, **u'continue': u'continue'**, **u'si nécessaire': u'non planifiée'**, **u'irrégulière': u'non planifiée'**, **u'non planifiée': u'non planifiée'**, **u'inconnue': u'non planifiée'**, **u'figée': u'figée'**}

Réinitialisation de la table ZZZ\_TDB\_METADONNEES et du champ tab\_maj\_statut dans ZZZ\_TDB\_TABLESDE :

arcpy.DeleteRows\_management(ZZZ\_TDB\_metadonnees)  
  
rows **=** arcpy.UpdateCursor(ZZZ\_TDB\_tablesde)  
**for** row **in** rows**:** row.setValue('tab\_maj\_statut', '')  
 rows.updateRow(row)  
**del** row, rows

Récupération de la date de dernière mise à jour dans le champ tab\_datemodif si tab\_meta\_id n’est pas une valeur nulle :

dicoSDE **=** {}  
rows **=** arcpy.SearchCursor(ZZZ\_TDB\_tablesde)  
**for** row **in** rows**:  
 if** row.getValue('tab\_meta\_id') **!=** None**:** dicoSDE[row.getValue('tab\_meta\_id')] **=** row.getValue('tab\_datemodif')  
  
**del** row, rows

Création d’un dictionnaire avec pour clé le nom de la fiche de métadonnées (= pkB72) et pour valeur la fréquence de mise à jour :

dicoDSITMeta **=** {}  
rows **=** arcpy.SearchCursor(tableDSITMeta)  
  
**for** row **in** rows**:** dicoDSITMeta[row.getValue('PKB72')] **=** row.getValue('FREQU')  
  
**del** row, rows

Création de la table ZZZ\_TDB\_METADONNEES avec remplissage des champs meta\_id, meta\_pkB72, meta\_maj\_freq et calcul de la meta\_maj\_date si possible (date butoir pour réaliser une mise à jour avant que la donnée ne soit considérée comme étant obsolète) :

rows **=** arcpy.InsertCursor(ZZZ\_TDB\_metadonnees)  
  
**for** i, metaKey **in** enumerate(dicoDSITMeta.keys())**:** row **=** rows.newRow()  
 row.setValue('meta\_id', i)  
 row.setValue('meta\_pkB72', metaKey)  
  
 **if** dicoDSITMeta[metaKey] **!=** None**:** *# si MàJ renseignée* row.setValue('meta\_maj\_freq', dicoDSITMeta[metaKey])  
 **if** isinstance(dicoPlanif[dicoDSITMeta[metaKey]], \_\_builtins\_\_.int )**:** *#si dans dicoPlanif* **if** metaKey **in** dicoSDE**:** *#si présent dans la table ZZZ\_TDB\_tablesde* **if** dicoSDE[metaKey] **!=** None**:** *#si a une date de modification  
 #date butoire pour prochaine maj* date\_butoire **=** dicoSDE[metaKey] **+** timedelta(days **=** dicoPlanif[dicoDSITMeta[metaKey]])  
 row.setValue('meta\_maj\_date', date\_butoire)  
  
 rows.insertRow(row)  
  
**del** row, rows

Pour le calcul, on ajoute à la date de mise à jour, renseignée par le référent, le laps de temps Glob().delai\_maj\_str défini dans param\_scan.py après l’avoir reconverti en date.

**Remarque :** Le champ meta\_maj\_date n’est pas reconverti au format string par la suite, comme il n’est pas utilisé dans les JSON. La manipulation à suivre si jamais ce champ devait intégrer le tableau de bord est la même que pour la conversion des dates de TIMESTAMP et LAST\_DDL\_TIME.

Mise à jour du champ tab\_maj\_statut en fonction des dates calculées et des fréquences de mise à jour :

* Création d’un dictionnaire de valeurs invalides, c’est-à-dire que ces valeurs ne permettent pas d’obtenir une date butoir :

rows **=** arcpy.SearchCursor(ZZZ\_TDB\_metadonnees)  
dicoStatutMaJ **=** {}  
maj\_invalide **=** {**u'si nécessaire'**, **u'maj invalide'**, **u'irrégulière'**, **u'non planifiée'**, **u'inconnue'**, None}

* Pour chaque ligne remontée, le script effectue une suite de vérification :
  + Si la fréquence de mise à jour est figée, alors le couple nom de la fiche : Figée est ajoutée au dictionnaire précédemment créé :
    - **if** row.getValue('meta\_maj\_freq') **== u'figée':** dicoStatutMaJ[row.getValue('meta\_PKB72')] **= u'Figée'**
  + Si elle est comprise dans le dictionnaire des fréquences invalide :
    - **if** row.getValue('meta\_maj\_freq') **in** maj\_invalide**:** dicoStatutMaJ[row.getValue('meta\_PKB72')] **=** None
  + Si elle est continue et qu’elle dispose d’une date de dernière modification récupérée dans JeuxDonnees\_Maj.py, il faut vérifier que l’écart entre cette date et la date du jour soit bien inférieur au Glob().delai\_maj\_str :
    - **if** row.getValue('meta\_maj\_freq') **== u'continue' and** last\_modif **!=** None**:** *# reconversion du Glob().delai\_maj\_str au format timedelta* delai\_maj **=** datetime.strptime(delai\_maj\_str, '%d')  
       delai\_maj **=** timedelta(days**=**delai\_maj.day, hours**=**delai\_maj.hour, minutes**=**delai\_maj.minute, \  
       seconds**=**delai\_maj.second, microseconds**=**delai\_maj.microsecond)  
        
       *# Calcul de l'écart entre la date de dernière modification et la date du jour* laps **=** datetime.strptime(last\_modif, '%Y-%m-%d %H:%M:%S') **-** now  
        
       **if** laps **<** delai\_maj**:** dicoStatutMaJ[row.getValue('meta\_PBK72')] **= u'-'  
       else:** dicoStatutMaJ[row.getValue('meta\_PKB72')] **= u'Mise à jour à faire'**
  + Si elle n’est dans aucun des cas précédents, mais qu’une date butoir a été calculée, le statut de la mise à jour est déterminé à partir de cette dernière :
    - date\_maj **=** row.getValue('meta\_maj\_date')  
      **if** date\_maj **!=** None**:  
       if** date\_maj **-** now **>** timedelta(0, 0, 0)**:** dicoStatutMaJ[row.getValue('meta\_PKB72')] **= u'-'  
       else:** dicoStatutMaJ[row.getValue('meta\_PKB72')] **= u'Mise à jour à faire'**

Pour tous les jeux de données qui n’ont pas été pris en compte par les tests antérieurs :

rows **=** arcpy.UpdateCursor(ZZZ\_TDB\_tablesde)  
**for** row **in** rows**:** nomJeu **=** row.getValue('tab\_nom')  
 **if** row.getValue('tab\_meta\_id') **in** dicoStatutMaJ**:** row.setValue('tab\_maj\_statut', dicoStatutMaJ[row.getValue('tab\_meta\_id')])  
  
 **elif** row.getValue('tab\_meta\_id') **not in** dicoStatutMaJ **and** row.getValue('tab\_meta\_id') **!=** None**:  
 if** row.getValue('tab\_type') **==** 'Vue'**:** *#Exception sur les vues* row.setValue('tab\_maj\_statut', **u'n/a'**)  
 **elif** '\_ATTACH' **in** nomJeu**:** *#Exception sur les pièces jointes* row.setValue('tab\_maj\_statut', **u'n/a'**)  
 **else:** row.setValue('tab\_maj\_statut', **u'Non planifiée'**)  
 rows.updateRow(row)  
  
**del** row, rows

## Calcul du pourcentage de données à jour (dans Datüm.py)

Après avoir inscrit dans les bases de données la totalité des informations que les scripts sur ont pu récupérer, FIN.py produit des fichiers JSON sur lesquels Datüm se base pour l’affichage des informations. Dans l’onglet d’accueil se trouve le donut sur le pourcentage de données à jour :

**class FormAccueil**(QWidget, Ui\_FormAccueil)**:**

[…]

**def loadDataDonuts**(self)**:**

[…]

*# Calcul pourcentage "Données à jour" selon le nombre de MD avec féquence màj définie*a **=** len([x **for** x **in** dataJeu['TAB\_MAJ\_STATUT'] **if** x **== u'Mise** \xe0 **jour** \xe0 **faire'**])  
self.jeu\_Maj **=** len([x **for** x **in** dataJeu['TAB\_MAJ\_STATUT'] **if** x **== u'-' or** x **== u'Figée'**])  
c **=** len([x **for** x **in** dataJeu['TAB\_MAJ\_STATUT'] **if** x **== u'Non planifi**\xe9**e'**])  
self.pctJeu\_Maj **=** float(self.jeu\_Maj) **/** (float(a) **+** float(self.jeu\_Maj) **+** float(c))

# Extension de communication avec les référents

Ce chapitre est une synthèse du document resume\_Relation\_Referents.docx

## Production des rapports

### Automatisé : Relation\_Referents.py

Définition des variables à partir de celles récupérée dans param\_scan.py et de la liste des dates :

**from** param\_scan **import \***JeuxSDE **=** Glob().ZZZ\_TDB\_tablesde  
baseSDE **=** Glob().baseSDEprod  
dirJson **=** Glob().dirJson  
nivQualData **=** Glob().NivQualData  
icons **=** Glob().dirIcons  
dirRapports **=** Glob().dirRapports  
envoi\_mails **=** Glob().envoi\_mails  
lstDates **=** [**"2018-07-03"**, **"2018-10-02"**, **"2019-01-08"**, **"2019-04-02"**, **"2019-07-02"**, **"2019-10-01"**, **"2020-01-07"**,  
 **"2020-04-01"**, **"2020-07-01"**, **"2020-10-01"**, **"2021-01-05"**, **"2021-04-01"**, **"2021-07-01"**, **"2021-10-01"**]

Si la date du jour est comprise dans les dates choisies par l’administrateur, alors le reste du script s’exécute. Sinon, un message type est inscrit dans le log pour que cette fin ne soit pas considérée comme une erreur.

**if** datetime.now().strftime(**"%Y-%m-%d"**) **in** lstDates**:**

* Création des dossiers pour trier les référents en fonction de leur affiliation. Chaque dossier reprend la date du jour transformée en string suivie d’une spécification :
* Définition des nouvelles variables et récupération du contenu de Niveau\_qualite\_donnees.xlsx
* Récupération par requête SQL d’informations sur les jeux de données, avec emploi du *to\_char((valeur), format)*pour convertir les dates en string et *to\_date((valeur), format)* pour convertir les entrées numériques en dates :

sql **= "SELECT TAB\_NOM, META\_TITRE, TAB\_CONTACT, TAB\_QUALITE, META\_MAJ\_FREQ,to\_char((META\_JEUDATEMODIF), 'YYYY/MM/DD'), to\_date(META\_MAJ\_FICHE, 'YYYY-MM'), META\_CHPVIDE, META\_EMAIL FROM SIG.ZZZ\_TDB\_tablesde, SIG.ZZZ\_TDB\_metadonnees WHERE TAB\_META\_ID=META\_PKB72 AND TAB\_NOM='{}'"**.format(row.getValue(**'TAB\_NOM'**))

listInfo **=** egdb\_conn.execute(sql)

* Petit script pour supprimer les caractères spéciaux et remplacer les caractères accentués par leur équivalent sans accent :

*# sup. chr pour éviter plantage dans nom fichier*list\_sc **=** [**'/'**, **'**\\**'**, **'\*'**, **'?'**, **'"'**, **'>'**, **'>'**, **'|'**]  
accents **=** {**'a':** [**u'à'**, **u'ã'**, **u'á'**, **u'â'**],  
 **'e':** [**u'é'**, **u'è'**, **u'ê'**, **u'ë'**],  
 **'i':** [**u'î'**, **u'ï'**],  
 **'u':** [**u'ù'**, **u'ü'**, **u'û'**],  
 **'o':** [**u'ô'**, **u'ö'**],  
 **'c':** [**u'ç'**]}  
nomContact **= ''**.join([i **if** i **not in** list\_sc **else '' for** i **in** listInfo[0][2]])  
**for** (char, accented\_chars) **in** accents.iteritems()**:  
 for** accented\_char **in** accented\_chars**:** nomContact **=** nomContact.replace(accented\_char, char)

* Si la qualité n’est pas null, on intègre toutes les informations relatives au jeu de données pour chaque occurrence de règle dans sa qualité (si J04 et J07, un jeu apparaît donc deux fois dans le rapport) :

regles **=** []  
 regles **=** listInfo[0][3].split(**','**)  
 **for** regle **in** regles**:  
 if** regle **in** dicoNivQualData.keys()**:** dicoEvent[row.getValue(**'TAB\_NOM'**) **+ '|' +** regle] **=** \  
 (nomContact, listInfo[0][1], dicoNivQualData[regle][0]**+' - '+**dicoNivQualData[regle][1],  
 dicoNivQualData[regle][2], dicoNivQualData[regle][3], dicoNivQualData[regle][4],  
 listInfo[0][4], listInfo[0][5], listInfo[0][6], listInfo[0][7], listInfo[0][8])  
 **else:** dicoInfo[row.getValue(**'TAB\_NOM'**)**+'|'+ 'JD'**] **=** \  
 (nomContact, listInfo[0][1], listInfo[0][8], **u"Jeu de données"**)  
**else:** dicoInfo[row.getValue(**'TAB\_NOM'**) **+'|'+ 'JD'**] **=** \  
 (nomContact, listInfo[0][1], listInfo[0][8], **u"Jeu de données"**)

* Pour les métadonnées, le principe est sensiblement le même, si ce n’est qu’il n’est plus nécessaire d’utiliser le tab\_nom comme identifiant pour les différentes conditions (listInfo[0][x] 🡪 listInfo[x]). Il est toujours obligatoire comme clé du dicoInfo.
* Suppression des doublons sur les métadonnées :

clesDoubles **=** list()  
**for** key **in** dicoEvent.keys()**:  
 if** key.split(**'|'**)[1] **in** dicoNivQualData.keys()**:  
 for** cle **in** dicoInfo.keys()**:  
 if** dicoInfo[cle][1] **==** dicoEvent[key][1] **and** cle.split(**'|'**)[1] **!= 'JD':** clesDoubles.append(cle)  
**for** cle **in** clesDoubles**:  
 if** cle **in** dicoInfo**:  
 del** dicoInfo[cle]

* Création

# Extension sur les Services Web

Toutes les informations mentionnées dans ce chapitre se trouvent sur le réseau interne de Brest métropole dans des fichiers de type JSON ou XML, accessibles par l’explorateur Windows

Services : [\\hcu234.dit.cb\v235\arcgisserver\config-store\services](file:///\\hcu234.dit.cb\v235\arcgisserver\config-store\services)

Rôles et utilisateurs : [\\hcu234.dit.cb\v235\arcgisserver\config-store\security](file:///\\hcu234.dit.cb\v235\arcgisserver\config-store\security)

Mais également présentes sur la page web d’ArcGIS Server Manager en parcourant les différents onglets de « Site (racine) > *Dossier* > *Nom* ».

[https://hcu234.dit.cb/admin/manager/index.html#](https://hcu234.dit.cb/admin/manager/index.html)

## Tables de bases de données et leur contenu

### ZZZ\_TDB\_SERVICEWEB

**SW\_ID :**

Clé d’identification générée par le script

**SW\_NOM (*technique*) :**

Nom du service tel qu’inscrit dans ArcGIS Server Manager, déterminé lors de la publication du service suivi du nom du dossier dans lequel il a été publié, séparé par une barre verticale ‘|’ (ou aussi appelée pipe). Cette valeur prend par défaut le nom du .mxd d’origine.

L’ajout du nom du dossier est devenu obligatoire lors de la production des JSON pour réussir à distinguer les différents services ayant un même nom par requête SQL.

**SW\_TITRE :**

Cette information n’est pas obligatoire mais peut-être renseignée lors de la publication sous ArcMap dans la partie ‘Description de l’élément’ dans le cadre ‘Résumé’.

Elle se retrouve dans ArcGIS Server Manager dans l’onglet « Description de l’élément > Récapitulatif »

Pour y accéder via le réseau interne, il faut trouver le fichier Dossier\Nom.TypeServer\esriinfo\ iteminfo.json et chercher le ‘snippet’ (ou ‘summary’, aucune différence n’ayant été trouvée entre ces deux sources).

**SW\_STATUT :** Démarré // Arrêté // En veille

Ce champ indique l’état actuel du service, c’est-à-dire s’il est Démarré ou Arrêté. Lorsqu’un service est indiqué comme étant En veille, cela signifie qu’il est arrêté mais qu’il doit effectivement être dans cet état.

Pour que cette distinction soit faite, il est nécessaire que la partie « Description de l’élément > Contraintes d'accès et d'utilisation » soit remplie d’un V. Cette valeur est ensuite récupérée dans le fichier iteminfo.json au niveau de ‘licenseInfo’.

**SW\_SOURCE :**

Le chemin d’accès au fichier (.mxd, .loc ou .tbx en fonction du type de service) ayant permis de produire le service tel qu’inscrit dans le fichier *Dossier*\*Nom*.*TypeServer*\esriinfo\manifest\ manifest.json au niveau de la balise 'onPremisePath'.

Elle correspond à celle indiquée dans « Général > Chemin » dans ArcGIS Server Manager.

A noter toutefois que la mise à jour de la source n’a pas d’impact sur le service et que dans certains cas exceptionnels, cette valeur peut-être (null).

**SW\_SOURCE\_PRESENTE :** OUI // NON // PC // BDD // Accès refusé

Indique si le chemin de la source qui a été récupéré mène effectivement au bon fichier.

PC : le fichier se situe sur un chemin commençant par C:/ ou D:/ et donc inaccessible mais impossible de déterminer s’il est réellement présent ou non

BDD : le fichier se situe sur un chemin comprenant un ‘.sde’ qui n’est pas compréhensible pour Windows, également impossible de savoir si présence ou non.

Accès refusé : pour le compte SIG, les volumes K:/, M:/ et [\\dit.cb\dfs\v505](file:///\\dit.cb\dfs\v505) sont inaccessibles.

**SW\_NB\_LIENS\_ABS :**

Ce chiffre correspond au nombre de couches dans le .mxd d’origine qui sont **actuellement** cassées, c’est-à-dire que le chemin menant au jeu de données sur lequel elle est basé n’est plus juste. Ceci n’implique pas obligatoirement l’absence de la couche dans le service. Par exemple, si le chemin a disparu après la publication, le service n’est pas impacté.

**SW\_PROCESSUS :** MapServer // GPServer // GeocodeServer

Le premier élément de ce champ est toujours le ‘type’ de service (présentés ci-dessus). Vient ensuite la liste des « Fonctionnalités » actives du service s’il est de type MapServer, séparés par un virgule et un espace ‘, ’.

FeatureServer // KmlServer // WFSServer // WMSServer

Ils sont tous présents dans le fichier *Dossier*\*Nom*.*TypeServer*\

*Nom.TypeServer*.json, dans la partie ‘extensions’ et renvoient la valeur ‘true’ s’ils sont activés.

**SW\_DROIT :** Editable // (null)

La fonctionnalité « Feature Access » présente des « Opérations autorisées » et des « Propriétés » qui se traduisent par la possibilité pour un utilisateur de modifier directement la donnée. Si l’une des quatre case - ‘Créer / ‘Requête’ / ‘Mettre à jour’ / ‘Supprimer’ - ou la case ‘Autoriser les mises à jour de la géométrie’ est cochée dans ArcGIS Server Manager, le service prend la valeur Editable.

Cela se traduit dans le *Nom.TypeServer*.json par :

* L’apparition des mots-clés – Create / Query / Update / Delete - en face des ‘capabilities’ de ‘extensions’ > ‘typeName’ : ‘FeatureServer’.
* Dans ses ‘properties’, ‘allowGeometryUpdates’ prend la valeur ‘true’.

**~~SW\_CREATEUR :~~**

Nom du propriétaire du fichier source indiqué dans ses propriétés, mais aucune option pour le récupérer n’a été concluante. Exemple : ‘HDVCUBD02 / avergne’

**SW\_SRC\_CREATION :**

Date à laquelle a été créé le fichier source, indiquée dans l’onglet ‘Détails’ des propriétés du fichier.

**SW\_SRC\_MAJ :**

Date de la dernière modification de la source, se situant au même endroit que la date de création.

**SW\_QUALITE :** S01 // S02 // S03 // S04 // S05 // S06 // S07

Ce champ contient la liste des alertes attribuées au service, séparées par une virgule et un espace ‘, ‘.

**SW\_QUAL\_IND :** 1 // 2 // 3

Le niveau de qualité en fonction de la plus haute criticité parmi les alertes remontées.

### ZZZ\_TDB\_USER\_ROLE

**USR\_ID :**

Clé d’identification générée par le script pour chaque nom d’utilisateur dans chaque rôle.

**USR\_USER\_ID :**

Clé d’identification générée par le script pour chaque nom d’utilisateur dans chaque rôle, classée en suivant l’ordre alphabétique des noms d’utilisateurs.

**USR\_USER\_NOM :**

Nom des utilisateurs apparaissant dans le dossier user-roles, composé de l’initiale du prénom suivie du nom. Ce nom apparaît autant de fois dans cette table qu’il y a de rôles faisant appel à lui dans le fichier JSON hdvcub02\_*nom*.

**USR\_ROLE\_ID :**

Clé d’identification générée par le script pour chaque rôle cité dans les fichiers d’utilisateur. Chaque occurrence d’un même rôle aura donc la même id.

**USR\_ROLE\_NOM :**

Nom du rôle cité dans les fichiers d’utilisateur après ‘roles’.

### ZZZ\_TDB\_ROLESERVICE (table relationnelle)

**ROSE\_ID :**

Clé d’identification générée par le script pour chaque occurrence d’un rôle dans les divers services publiés (*si ‘admin’ est présent dans 3 services différents, il y aura donc 3 ‘ROSE\_ID’*).

**ROSE \_SW\_ID:**

Clé d’identification du service web, récupérée dans la table ZZZ\_TDB\_SERVICE, faisant appel au rôle ayant l’id du champ ROSE\_USR\_ROLE\_ID.

**ROSE\_USR\_ROLE\_ID:**

Clé d’identification du rôle récupérée dans la table ZZZ\_TDB\_USER\_ROLE.

### ZZZ\_TDB\_COUCHE

**COU\_ID :**

Clé d’identification générée par le script pour chaque couche présente dans les sources des services publiés.

**COU\_NOM:**

Nom de chacune de ces couches tel qu’enregistré dans le .mxd source.

### ZZZ\_TDB\_COUCHESERVICE (table relationnelle)

**COSW\_ID :**

Clé d’identification générée par le script pour chaque couche présente dans les sources des services publiés et dont le jeu de données d’origine est également présent dans la table ZZZ\_TDB\_TABLESDE.

**COSW \_COU\_ID:**

Clé d’identification de la couche récupérée dans la table ZZZ\_TDB\_COUCHE.

**COSW \_TAB\_ID:**

Clé d’identification du jeu de données d’origine de la couche récupérée dans la table ZZZ\_TDB\_TABLESDE.

**COSW \_SW\_ID:**

Clé d’identification du service web dont la source est le .mxd où se situe la couche, récupérée dans la table ZZZ\_TDB\_SERVICEWEB.

## Scripts Services\_[…].py, param\_scan.py et FIN.py

### param\_scan.py

Pour le bon déroulement des autres scans, plusieurs variables ont été ajoutées :

Celles pointant vers les tables :

* self.ZZZ\_TDB\_serviceweb = r'H:\etu\sde\Connexions\_bases\SIG\_prod\_SIG.sde\SIG.ZZZ\_TDB\_serviceweb'
* self.ZZZ\_TDB\_couche = r'H:\etu\sde\Connexions\_bases\SIG\_prod\_SIG.sde\SIG.ZZZ\_TDB\_couche'
* self.ZZZ\_TDB\_coucheservice = r'H:\etu\sde\Connexions\_bases\SIG\_prod\_SIG.sde\SIG.ZZZ\_TDB\_coucheservice'
* self.ZZZ\_TDB\_user\_role = r'H:\etu\sde\Connexions\_bases\SIG\_prod\_SIG.sde\SIG.ZZZ\_TDB\_user\_role'
* self.ZZZ\_TDB\_roleservice = r'H:\etu\sde\Connexions\_bases\SIG\_prod\_SIG.sde\SIG.ZZZ\_TDB\_roleservice'

Celles pointant vers les dossiers de services :

* self.dirServices = r'\\hcu234.dit.cb\v235\arcgisserver\config-store\services'
* self.dirRoles = r'\\hcu234.dit.cb\v235\arcgisserver\config-store\security'

### Services\_Bases.py

Ce script récupère la liste des services web qui sont présents dans les dossiers de l’arborescence [\\hcu234.dit.cb\v235\arcgisserver\config-store\services](file:///\\hcu234.dit.cb\v235\arcgisserver\config-store\services) en excluant les dossiers System et Utilities qui sont propres à ArcGIS Server Manager.

Pour ce faire, il parcourt la liste des fichiers présents dans le dossier du service, ouvre *Nom.TypeServer*.json puis récupère les valeurs suivant ‘serviceName’ et ‘configuredState’ présentes dans ce fichier.

Après suppression par requête SQL dans la table ZZZ\_TDB\_SERVICE des lignes dont le nom technique (*Nom | Dossier*) n’est pas présent dans la liste des noms techniques récupérés ainsi, le script enregistre dans cette table les nouveaux services en remplissant les champs SW\_ID, SW\_NOM et SW\_STATUT.

Précision : à ce stade du scan, le statut d’un service ne peut être que STARTED ou STOPPED.

### Services\_Infos.py

Ce scan enregistre toutes les informations relatives au service Web qui sont enregistrées dans les divers .json du dossier, à partir des noms techniques produits dans le précédent.

Dans *nom.ServerType*.json :

* Le type de service : ‘type’  soit MapServer // GPServer // GeocodeServer, la première entrée di dictionnaire remplissant le champ SW\_PROCESSUS.
* Les processus : dans la partie ‘extensions’, si une occurrence de fonctionnalité affiche ‘true’ pour ‘enabled’, « , ‘typeName’ » est ajouté au dictionnaire des processus.
* Les droits : si le processus ‘FeatureServer’ affiche l’un des mots-clés – Create / Query / Update / Delete - en face des ‘capabilities’ ou bien si dans ses ‘properties’, ‘allowGeometryUpdates’ prend la valeur ‘true’, le service devient Editable.

Dans esriinfo\ *iteminfo*.json :

* Le titre du service : si renseigné, il se trouve après ‘snippet’
* La veille : si ‘licenseInfo’ n’affiche qu’un V, c’est l’indication qui a été choisie pour signifier qu’il était normal que le service en question soit STOPPED. SW\_STATUT prend alors la valeur En veille.

Dans esriinfo\manifest\manifest.json (*ce dossier peut parfois être absent*):

* La source : dans les ‘resources’, le mot-clé ‘onPremisePath’ est suivi du chemin menant au fichier ayant permis la publication du service. Il s’agit soit un .mxd, soit un .tbx ou encore d’un .loc

### Services\_SourceInfos.py

Ce script détermine si le chemin Windows de la source mène toujours au bon fichier ou s’il a disparu.

Si elle est toujours présente, il récupère la date de création de cette source ainsi que la date de sa dernière modification. Ce sont toutes deux des valeurs temporelles dont la forme est paramétrable par l’administrateur (actuellement au format « Y%/%m/%d ».

### Services\_Couches.py

Ce scan récupère la liste des couches et des jeux de données associés présents dans la source s’il s’agit d’un fichier ArcMap puis les place dans ZZZ\_TDB\_COUCHE. Il les compare ensuite aux jeux de données présent dans ZZZ\_TDB\_TABLESDE et établit la relation entre les différents id (SW\_ID, COU\_ID et TAB\_ID) dans la table ZZZ\_TDB\_COUCHESERVICE.

Si un lien vers un jeu de données n’est est cassé, c’est-à-dire que la couche n’a plus de source, l’information est remonté pour le service dans le champ SW\_NB\_LIEN\_ABS.

### Services\_UserRole.py

Ce script enregistre tous les rôles et utilisateurs qui leurs sont liés en parcourant les différents .json du dossier users-roles. Le nom de l’utilisateur suit la balise ‘username’ tandis que celui du rôle se trouve après ‘roles’.

La clé primaire est id est créé pour chaque couple utilisateur / rôle, mais chaque utilisateur et chaque rôle disposent d’un identifiant unique supplémentaire dans ZZZ\_TDB\_USER\_ROLE.

Les lignes de cette table sont ensuite comparées aux informations recueillies dans les fichiers nommés *Dossier.Nom.TypeServer*.sec qui sont associés aux services éponymes. Ils se situent à la racine des dossiers de services (ex : public, secure). Il ne reste alors plus qu’à remplir la table ZZZ\_TDB\_ROLESERVICE avec les identifiants des users-roles et des services.

### Services\_Qualite.py

Ce dernier scan établit le niveau de qualité de chaque service en fonction des alertes définies dans le fichier Excel Niveaux\_qualite.xls se trouvant dans data. Chaque valeur est séparée par une virgule dans le champ SW\_QUALITE et l’alerte de plus haut niveau remontée donne l’indicateur correspondant au champ SW\_QUAL\_IND.

### FIN.py

La production des fichiers JSON sur les services web est identique à celle des autres éléments analysés par Datüm.

Dans un premier temps, le programme récupère toutes les informations contenues dans la table ZZZ\_TDB\_SERVICEWEB à l’aide d’une requête SQL. Les champs de type date suivent un traitement particulier car ils ne sont pas lus correctement par JSON, il faut donc les convertir en string avant de les ajouter au dictionnaire sur les services. Ce même dictionnaire est ensuite simplement exporté dans le format souhaité, nommé service\_tableau.json.

Une suite de requête et d’exports permet de la même manière de créer les fichiers contenants les ensembles Service = [couches] ; Service = [Jeux] ; Jeu = [Services] ; Service = [Utilisateurs] ; Rôle = [Utilisateurs] ; Service = [Rôles] et enfin, Rôle = [Services].

Les chiffres relatifs aux services web sont ajoutés à histo\_global.json avec une valeur inscrite manuellement de 0 pour tous les mois précédents le début des scans sur ces éléments. Les évolutions de qualités des services ont également été ajoutées au JSON récapitulatif.

## Modification et production graphique (Qt Designer)

### form\_accueil.ui

Pour cette interface, txt\_service (*QLabel*) et widget\_Service (*QWidget*) sont les seuls éléments à avoir été ajoutés. L’espace nécessaire à leur intégration a été gagné en diminuant la taille des *horizontalSpacer* situés sur les côtés de la fenêtre.

### form\_sde.ui

Dans le tableau des jeux de données, une nouvelle colonne contenant un hyperlien vers une fenêtre de type form\_liste.ui a été ajoutée. Elle affiche lors de l’ouverture la liste des services utilisant le jeu de données.

### form\_service.ui

Cet onglet reprend la structure de l’onglet sur les fichiers Lyr, avec quelques ajouts provenant de l’onglet sur les jeux de données. Voici la liste des éléments principaux et leur fonction :

**lst\_service** (*QListWidget*) :

Ce cadre de visualisation contient l’ensemble des services présents dans le dossier services ou celui affiché dans cmb\_dossier (*QComboBox*), tandis que le *QLineEdit* permet de les filtrer selon leur nom. Le btn\_dossier (*QPushButton*) ne s’active que si une entité est sélectionnée car il se charge d’ouvrir son emplacement dans un explorateur Windows. Enfin, le lbl\_nb\_service (*QLabel*) a pour fonction d’afficher le nombre de services présents dans le dossier lors du dernier scan effectué.

**tree\_service** (*QTreeView)*:

Si le service sélectionné précédemment contient des couches, ce cadre en affiche la liste, suivie de la liste des jeux de données ayant servi à les produire. Bien que ces éléments soient éditables, aucune modification n’est enregistrée et il suffit de changer de sélection puis de revenir sur ce service pour annuler une erreur de manipulation.

La première ligne rappelle le nom de l’entité et précise à sa suite qu’elle ne dispose pas de couches si tel est le cas.

**tbl\_entete** (*QTableWidget)*:

Cet espace accueille le tableau récapitulatif de l’état des services web selon les 4 valeurs de criticité des alertes remontées (None, 1, 2, 3) avec une brève description, les icônes correspondantes et la quantité de services concernés.

**cmb\_role** (*QComboBox)*:

Cette liste déroulante comporte les différents rôles enregistrés par les scans et ouvre une nouvelle fenêtre (définie par form\_liste.ui) qui contient la liste des services concernés.

**tbl\_service** (*QTableWidget)*:

Cette table permet de rendre lisible aisément le contenu de service\_tableau.json, qui peut être classé selon chaque entête. De plus, elle est dotée de la possibilité de filtrer les lignes en fonction du nom grâce au txt\_filtre (*QLineEdit*) ou en fonction de l’alerte choisie dans cmb\_filtre (*QComboBox*). Si nécessaire, le btn\_qual (*QPushButton*) ouvre une nouvelle fenêtre rappelant la liste de ces alertes en les classant par niveau d’importance.

La colonne Role-Users de ce tableau est constituée de case vide (si None) ou bien d’hyperlien oui qui renvoie une fenêtre de type form\_role\_user.ui

### form\_role\_user.ui

Calquée sur le fonctionnement de form\_domaine.ui, le lbl\_role (*QLabel*) présente une phrase simple suivie du nom du service. La lst\_role (*QListWidget*) qui lui succède affiche la liste des rôles qui lui sont attribués. De la même manière, le lbl\_user (*QLabel*) et la lst\_user (*QListWidget*) présentent le sinformations relatives aux utilisateurs liés au rôle sélectionné au-dessus.

### Production des .py à partir des .ui

Python n’étant pas capable de lire les fichiers de type .ui, il faut les convertir en fichiers de forme python à l’aide du script compile.py situé dans le dossier form. Pour se faire, et comme précisé dans les premières lignes du script, il faut lancer **OSGeo4W shell** (normalement fourni avec **QGIS**), ce qui ouvre une invite de commande. Après quelques instants, les lignes GDAL 1.11.3, released 2015/09/16 et C :\ apparaissent.

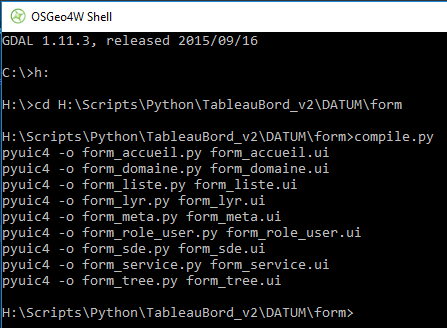
Il faut alors indiquer le volume que l’on souhaite atteindre, puis entrer cd suivi du chemin du dossier contenant les formes à convertir :

cd H:\Scripts\Python\TableauBord\_v2\DATUM\form

On peut alors entrer le nom du fichier à exécuter, soit compile.py. L’ordinateur renvoie alors la liste des fichiers converti en suivant cette présentation :

module–fonctionfichier\_produit.py fichier\_source.ui

Lorsqu’il a terminé, le programme rend la main à l’utilisateur en affichant le chemin dernièrement renseigné.



## L’intégration à l’IHM (= Datum.py)

Attention : Les éléments de ce chapitre apparaissent dans l’ordre des lignes du script, ce qui n’est pas forcément synonyme d’ordre chronologique d’opération.

Dans un premier temps, il a fallu intégrer l’onglet nouvellement conçu aux paramètres de mise en forme de Datüm en ajoutant ces deux lignes à param\_form.py :

**from** form\_service **import \*  
from** form\_role\_user **import \***

Le script Datum.py est constitué d’une fenêtre principale (*QMainWindow*) définie par une *class* faisant appel à quatre autres *class*, c’est-à-dire les différents onglets.

**class FenPrincipale**(QMainWindow)**:**

Pour ajouter un onglet, il faut donc produire une nouvelle sous-fenêtre basée sur une nouvelle *class* qu’il faut concevoir dans ce même script. Voici les quelques lignes ajoutées dans FenPrincipale(QMainWindow):

form4 **=** FormService()

[…]

sousFenetre4 **=** QMdiSubWindow()  
sousFenetre4 **=** self.zoneCentrale.addSubWindow(form4)  
self.zoneCentrale.setActiveSubWindow(sousFenetre4)

### L’onglet ‘Accueil’

**class FormAccueil**(QWidget, Ui\_FormAccueil)**:**

**def \_\_init\_\_**(self)**:**

Afin d’afficher de nouveaux donuts avec les informations relatives aux services web dans cet onglet, il faut tout d’abord ajouter une couleur (ici #DDB1D3) à la variable color. A noter que la position dans la liste a son importance.

color **=** ['#FFCBB5', '#9BA3FF', '#89CC68', '#DDB1D3' ]

Les lignes ci-après sont responsables de la production géométrique des donuts à partir de variables qui elles sont définies par la fonction loadDataDonuts(self).

*# définition donuts sur les services web*layout **=** QHBoxLayout(self.widget\_Service)  
layout.addWidget(  
 self.make\_pie([self.ptcSw\_actif, 1 **-** self.ptcSw\_actif], **u'{}%'**.format(int(self.ptcSw\_actif **\*** 100)), \  
 **u'Services**\n**actifs (' +** str(self.sw\_actif) **+** ')', color[3]))  
layout.addWidget(  
 self.make\_pie([self.ptcSw\_public, 1 **-** self.ptcSw\_public], **u'{}%'**.format(int(self.ptcSw\_public **\*** 100)), \  
 **u'Services**\n**publics (' +** str(self.sw\_public) **+** ')', color[3]))  
layout.addWidget(  
 self.make\_pie([self.ptcSw\_Jeu, 1 **-** self.ptcSw\_Jeu], **u'{}%'**.format(int(self.ptcSw\_Jeu **\*** 100)), **u'Jeux de données**\n**publiés (' +** str(self.sw\_Jeu) **+** ')', color[3]))

Le tout en n’oubliant pas de spécifier le titre de notre quatrième jeu de donuts :

self.txt\_service.setText(str(self.totalSw)**+ u' services web'**)

**def loadDataDonuts**(self)**:**

Comme précisé précédemment, toutes les valeurs numériques intervenant dans la conception des donuts sont calculées par cette fonction. Voici la partie concernant les services web :

*# ------- DONUTS SERVICES WEB -----------  
  
# Ouverture fichier services web*js **=** open(dirJson **+** '//service\_tableau.json')  
dataSw **=** json.load(js)  
js.close()  
  
js **=** open(self.dirJson **+** '//service\_dicoJeu\_Sw.json')  
self.dicoJeu\_Sw **=** json.load(js)  
js.close()  
  
*# Calcul nombre total de services web*self.totalSw **=** len([x **for** x **in** dataSw['SW\_QUALITE']])  
  
*# Calcul pourcentage "Services actifs"*self.sw\_actif **=** self.totalSw **-** len([x **for** x **in** dataSw['SW\_STATUT'] **if** x **== u'Arrêté'**])  
self.ptcSw\_actif **=** float(self.sw\_actif) **/** float(self.totalSw)  
  
*# Calcul pourcentage "Services publics"*self.sw\_public **=** self.totalSw **-** len([x **for** x **in** dataSw['SW\_NOM'] **if** x.split('|')[1] **!=** 'public'])  
self.ptcSw\_public **=** float(self.sw\_public) **/** float(self.totalSw)  
  
*# Calcul pourcentage "Nombre de jeux de données publiés"*self.sw\_Jeu **=** len(self.dicoJeu\_Sw.keys())  
self.ptcSw\_Jeu **=** float(self.sw\_Jeu) **/** float(self.totalJeu)

**def graphHisto**(self, *color*)**:**

Bien que les données sur l’historique des services web soient effectivement présentes dans histo\_global.json, il faut ajouter quelques variables dans cette fonction et modifier le titre :

histo\_Service **=** []

[…]

**for** i, key **in** enumerate(sorted(dicoHisto.keys(), reverse **=** True))**:  
 if** i **<** 24**:** *# valeurs sur 2 ans*

[…]

histo\_Service.append(dicoHisto[key][3])

[…]

histo\_Service.reverse()

[…]

plt.plot(x, histo\_Service,"-", linewidth**=**2, color**=***color*[3])

[…]

plt.title(**u"Historique (Jeux - Métadonnées - Lyr - Services)"**)

### L’onglet ‘Jeux de données’

**class FormSde**(QWidget, Ui\_FormSDE)**:**

Puisque l’apparition des services web dans Datüm permet de connaître la liste de ceux utilisant un jeu de données, il est nécessaire d’ajouter une colonne au tableau principal de cet onglet.

**def \_\_init\_\_**(self)**:**

* Import du JSON contenant les informations Jeu = [Services web] :

js **=**open(self.dirJson **+** "//service\_dicoJeu\_Sw.json")  
self.dicoJeu\_Sw **=** json.load(js)  
js.close()

* Ajout du champ et de l’entête de la nouvelle colonne (la position est libre de choix):

self.header **=** [('TAB\_NOM', **u'Jeu de données'**), […],('Service', **u'Service'**)]

**def loadDataTableau**(self)**:**

Toutefois, les données destinées à remplir ce tableau peuvent avoir des spécificités et il faut établir leur ordre d’apparition, ceci étant le rôle de cette fonction.

* Spécification du comportement à adopter pour la colonne des services web :

**elif** champ **==** 'Service'**:** self.posService **=** a *#position dans la table* self.dicoService **=** {} *#création d'un dico pour lister les services (dans hyperlink)* **for** i **in** range(len(loadData[champs[0]]))**:  
 if** loadData['TAB\_NOM'][i] **in** self.dicoJeu\_Sw.keys()**:** new\_item **=** QTableWidgetItem('liste')  
 new\_item.setTextColor(QColor(0,0,204,255))  
 new\_item.setFont(font)  
 table.setItem(i,a, new\_item)  
 self.dicoService[loadData['TAB\_NOM'][i]] **=** self.dicoJeu\_Sw[loadData['TAB\_NOM'][i]]  
 **else:** table.setItem(i, a, QTableWidgetItem(''))

Rappel : pour le moment, si un jeu de données est présent dans les clés du dicoJeu\_Sw, il prendra simplement la valeur liste dans la colonne ‘Service’.

**def hyperlink**(self)**:**

Pour que l’ouverture de la fenêtre s’opère lors d’un clic sur la case, il faut qu’une série d’action soit définie dans cette fonction.

*# définition de l'hyperlien sur les services***if** self.tbl\_sde.currentColumn() **==** self.posService**:  
 if** valCell **==** 'liste'**:** id\_jeu **=** self.tbl\_sde.item(self.tbl\_sde.currentRow(), 0).text()  
 fenetreListe **=** FormListe(id\_jeu, self.dicoService[str(id\_jeu)], **u'Services web associés au jeu de'  
 u'données '**) *# envoi nomJeu, liste, texte* fenetreListe.exec\_()

### L’onglet ‘Services web’

Cette partie explique en détails le fonctionnement d’un onglet spécifique, mais faisant appel à un maximum de fonctions et méthodes observables dans le script. Cela devrait permettre de produire de futures améliorations ou onglets plus aisément.

**class FormService**(QWidget, Ui\_FormService)**:**

**def \_\_init\_\_**(self)**:**

* Définition de la classe principale du FromService (*QWidget*) :

QWidget.\_\_init\_\_(self)  
self.setupUi(self)

* Import des fichiers JSON sous la forme de dictionnaires :

*#cablage aux fichiers Json*self.dirJson **=** Glob().dirJson  
  
js **=**open(self.dirJson **+** '//service\_tableau.json')  
self.tabledata **=** json.load(js)  
js.close()

* Création d’une liste de nom|dossier des services à partir du dictionnaire :

*#Init liste des services*self.listeSw **=** self.tabledata['SW\_NOM']

* Attribution d’une icône du dossier icons à btn\_dossier (*QPushButton*) :

*#Icone du dossier*iconDos **=** QtGui.QPixmap(os.path.dirname(os.path.abspath(\_\_file\_\_)) **+** '\icons\dossier.png')  
self.btn\_dossier.setIcon(QtGui.QIcon(iconDos))

* Définition de la liste des éléments de cmb\_filtre (*QComboBox*) avec parcours d’une liste de fichiers dans l’explorateur Windows.

*#initialisation de la combo box "cmb\_filtre"*self.listFiltre **=** []  
self.listFiltre.append('Tous')  
self.cmb\_filtre.addItem('Tous')  
  
**for** i **in** self.defQual.keys()**:  
 if** self.defQual[i][0] **== u'Services Web':** self.listFiltre.append(str(i))  
 val **=** self.defQual[i][1]  
 self.cmb\_filtre.addItem(val)  
  
*# Récupération de la liste des dossiers de services web*self.listDossier **=** ['Tous']  
path **=** r'\\hcu234.dit.cb\v235\arcgisserver\config-store\services'  
**for** dossier **in** os.listdir(path)**:  
 if** dossier **!=** 'System' **and** dossier **!=** 'Utilities'**:** *# Exclusion des dossiers spécifiques à ESRI* type **=** path **+** '\\' **+** dossier  
 **if** os.path.isdir(type) **is** True**:** self.listDossier.append(dossier)  
self.cmb\_dossier.addItems(self.listDossier)

* Choix de l’ordre des colonnes, de leur nom et de l’alias sous lequel elles apparaîtront (le nom est volontairement identique à la clé du dictionnaire qui contient les données destinées à remplir cette colonne).

*#Définition ordre des colonnes*self.header **=** [('SW\_NOM', **u'Nom'**), ('SW\_NOM', **u'Dossier'**), ('SW\_STATUT', **u'Statut'**),('SW\_QUAL\_IND', **u''**), \  
 ('SW\_TITRE', **u'Titre du service'**), ('SW\_SRC\_PRESENTE', **u'Présence Source'**), \  
 ('SW\_SOURCE', **u'Nom de la source'**), ('SW\_NB\_LIEN\_ABS', **u'Couches sans lien'**), \  
 ('SW\_DROIT', **u'Droits'**), ('SW\_SRC\_CREATION', **u'Création Source'**), ('SW\_SRC\_MAJ', **u'MàJ Source'**), \  
 ('SW\_PROCESSUS', **u'Liste des processus'**), ('Roles-Users', **u'Roles-Users'**)]

* Appel des fonctions de chargement de données et définition des connections.

*#Chargement des données*self.loadWidget()  
self.loadEnteteTableau()  
self.loadDataTableau()  
self.loadLstCoucheJeu()

1. La première variable dans la parenthèse est nommée d’après le nom de l’objet tel qu’inscrit dans Qt Designer.
2. Le SIGNAL est spécifique à une liste de signaux associée à l’objet en question
3. Le dernier élément est une fonction définie plus loin dans le script OU parfois une fonction d’une liste de fonctions par défaut associée à l’objet (= SLOT).

*#Définition des connecteurs*self.connect(self.cmb\_dossier, SIGNAL("currentIndexChanged(int)"), self.filtreDossier)  
self.connect(self.lineEdit, SIGNAL("textChanged(QString)"), self.loadListeService)  
self.connect(self.btn\_dossier, SIGNAL("clicked()"), self.affDossier)  
self.connect(self.btn\_qual, SIGNAL("clicked()"), self.indicName)  
self.connect(self.lst\_service, SIGNAL("itemSelectionChanged()"), self.affInfo)  
self.connect(self.cmb\_role, SIGNAL("activated(int)"), self.role)  
self.connect(self.cmb\_filtre, SIGNAL("currentIndexChanged(int)"), self.loadDataTableau)  
self.connect(self.txt\_filtre, SIGNAL("textChanged(QString)"), self.loadDataTableau)  
self.connect(self.tbl\_service, SIGNAL("itemSelectionChanged()"), self.hyperlink)

**def loadWidget**(self)**:**

Définition de la fonction indiquant le nombre de services dans le dossier public et dans le dossier secure.

**def filtreDossier**(self)**:**

Définition de la liste des dossiers apparaissant dans lst\_service (*QListWidget*), en fonction de l’objet sélectionné dans cmb\_dossier (*QComboBox*).

**def loadListeService**(self)**:**

Chargement de la liste des services dans le dossier sélectionné dans lst\_service avec affichage du nombre total d’entités présentes. Mise en place de l’activation conditionnelle du btn\_dossier (*QPushButton*).

**def affDossier**(self)**:**

Définition du comportement du btn\_dossier lors d’un clic (*ouverture du dossier du service dans l’explorateur Windows*)

**def affInfo**(self)**:**

Chargement de la liste des couches et des jeux de données associés dans tree\_service (*QTreeView*) en fonction du service sélectionné dans lst\_service. Si la source de cet objet ne dispose pas de couches, un message le précisant apparaît.

**def role**(self)**:**

Ouverture du pop-up avec la liste des services soumis au rôle lors d’un clic sur l’un d’entre eux dans cmb\_role (*QComboBox*).

**def loadEnteteTableau**(self)**:**

Création du tableau récapitulatif des niveaux d’alerte, de l’icône correspondante et rappel sur leur importance. Calcul du nombre de services concernés par chacun.

**def loadDataTableau**(self)**:**

Création du tableau principal de cet onglet. Mise en place des filtres selon le nom (caractères entrés dans txt\_filtre (*QLineEdit*) et également selon l’alerte affichée dans cmb\_filtre.

Spécification du contenu et/ou de son intégration de certaines colonnes (= champs), notamment pour les dates, les valeurs numériques et les hyperliens.

**def hyperlink**(self)**:**

Définition du comportement des cases ayant du contenu nécessitant une fenêtre particulière pour l’affichage de toutes les informations, en fonction de la colonne dans laquelle elle se trouve.

**def indicName**(self)**:**

Gestion du contenu de rappel sur les règles de qualité qui apparaissent dans une nouvelle fenêtre, FormIndicateur, classées par niveau de criticité.

### La fenêtre ‘Rôles et Utilisateurs’

**class FormRoleUser**(QDialog, Ui\_FormRoleUser)**:**

**def \_\_init\_\_**(self, *nomRole*, *dicoRoleUser*)**:**

Charge le fichier service\_dicoRole\_User.json contenant les couples Rôle = [Utilisateurs] qui sont attribués au service, dont le nom est rappelé dans la première ligne de la fenêtre. Ils apparaissent dans le cadre supérieur.

**def loadListUser**(self)**:**

A partir du même fichier, cette fonction remplit le cadre inférieur avec la liste des utilisateurs ayant accès au rôle sélectionné.