🛠️ De l'utilisation de Talend pour l'OpenData de l'Assemblée Nationale

0. Introduction et prérequis

Dans la suite du document, nous proposons des exercices qui sont liés les uns des autres.
Il est donc nécessaire de suivre dans l'ordre le document car certains exercices sont conditionnés aux précédents.
Nous proposons le traitement de données issu de l'Open Data de l'Assemblée Nationale, en vue de développer une API permettant d'obtenir les informations relatives à un député. Nous réaliserons également une route permettant à un utilisateur de voter sur un texte de loi donné.

Pour la réalisation des exercices, il sera nécessaire d'avoir :

- **Talend** en version 7.3.1 pour faire des traitements informatiques
    - dont **activemq**
    - dont **apache camel**
- **Notepad++** pour visualiser et éditer des fichiers
- **Dbeaver** pour interagir avec une base de données
- **PostgreSQL** pour disposer d'une base de données
- **Postman** pour interagir avec des API

- **Git** pour le développement collaboratif (Éventuellement un compte **Bitbucket** ou **Github**)
- Connexion internet
- Une compréhension basique de l'anglais technique
- Une compréhension basique de la science des données
- Une compréhension basique de l'informatique décisionnelle
- Une compréhension basique de la Politique

1. Présentation

1.1 Présentation de Talend ESB

1.1.1 Présentation du principe ESB

L'Enterprise Service Bus (ESB) est un modèle d'architecture logicielle qui prend en charge l'échange de données en temps réel entre des applications disparates. Les grandes entreprises disposent de plusieurs applications qui exécutent diverses fonctions en utilisant des modèles de données, des protocoles et des restrictions de sécurité différents. L'ESB facilite l'intégration des applications en effectuant des opérations telles que la transformation des données, la conversion des protocoles et le routage des messages. Les applications transmettent les données pertinentes à l'ESB, qui les convertit et les transmet à d'autres applications qui en ont besoin.

Talend ESB (Enterprise Service Bus), développé par la société Talend, permet le développement de bus applicatif de type middleware messaging et facilite grandement l'exposition de données via des API (Rest & SOAP)

1.1.2 Présentation du principe ETL

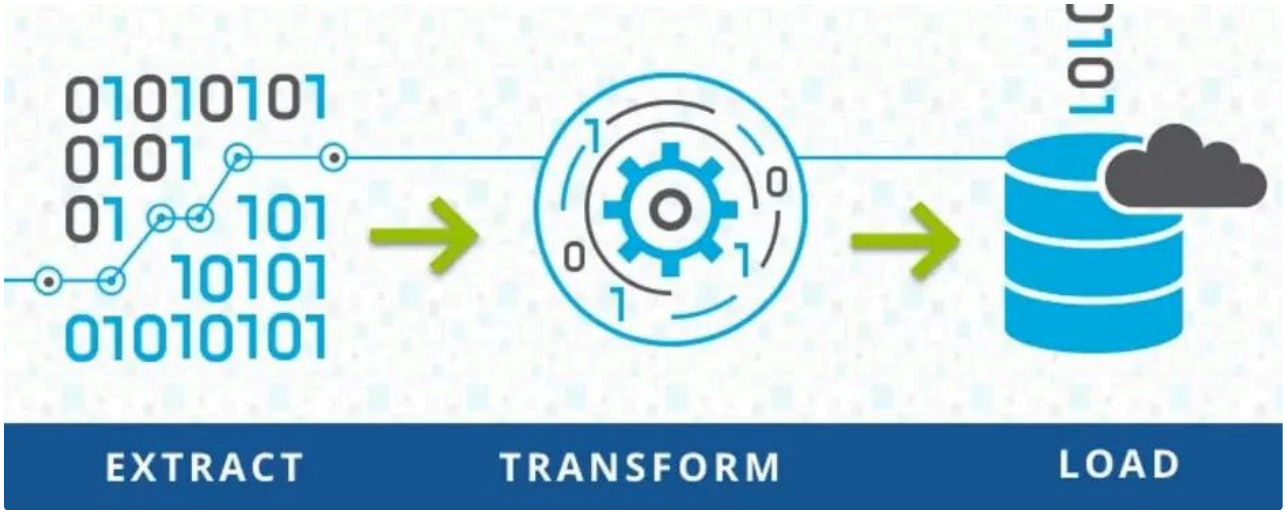
C'est un concept signifiant le chargement de données d'un point A vers un point B ou les données de A sont éventuellement transformé afin d'être chargés dans B.
Un logiciel ETL (Extract, Transform, Load) permet d'extraire des données brutes depuis une base de données, pour ensuite les restructurer, et enfin les charger.

Les premiers ETL ont fait leur apparition dans les années 1970, mais ont beaucoup évolué pour répondre aux nouveaux besoins liés à l'essor du Cloud, des SaaS (logiciels en tant que service) et du Big Data.
Désormais, les ETL doivent permettre l'ingestion en temps réel, l'enrichissement de données, la prise en charge de milliards de transactions. Ils prennent aussi en charge les données structurées ou non structurées en provenance de sources sur site ou sur le Cloud.

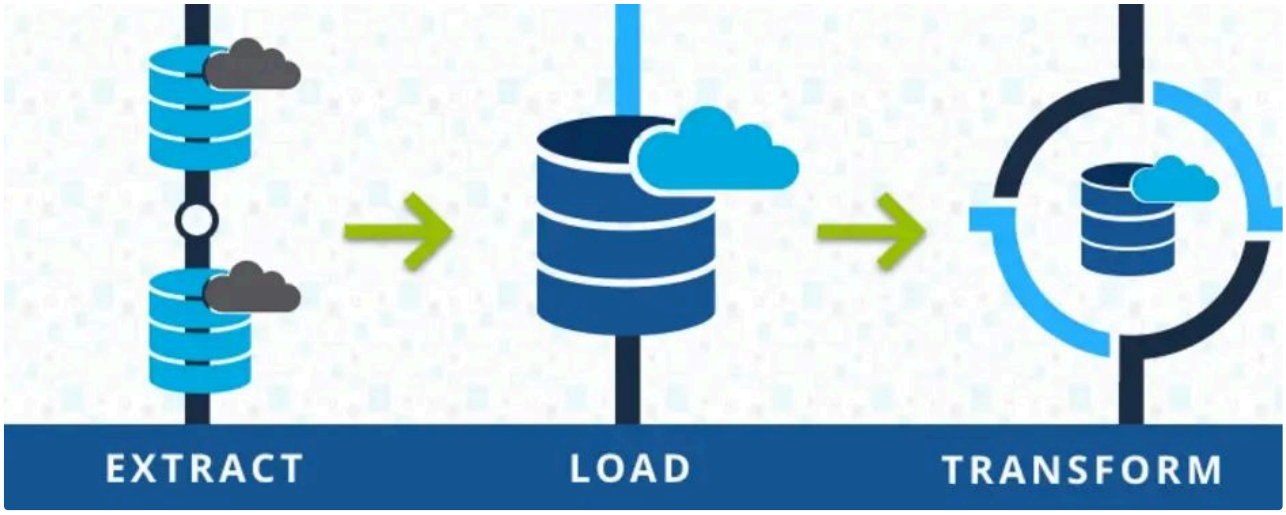
De même, ces plateformes doivent désormais être scalables, flexibles, résistantes aux pannes, et sécurisées.

La différence entre l'ETL et l'ELT réside dans le fait que les données sont transformées en informations décisionnelles et dans la quantité de données conservée dans les entrepôts.

- L'ETL (Extract/Transform/Load) est une approche d'intégration qui recueille des informations auprès de sources distantes, les transforme en formats et styles définis, puis les charge dans des bases de données, sources de données ou entrepôts.



- L'ELT (Extract/Load/Tansform) extrait également des données à partir d'une ou plusieurs sources distantes, mais les charge ensuite dans l'entrepôt de données cible sans changement de format. Dans un processus ELT, la transformation des données s'effectue au sein de la base de données cible. L'ELT nécessite moins de sources distantes, uniquement leurs données brutes et non préparées.



Les deux approches sont viables, mais les décideurs informatiques, lorsqu'ils créent une architecture de données, doivent prendre en compte les capacités internes et l'impact croissant des technologies Cloud.

Talend ESB (Enterprise Service Bus) est donc également un logiciel ETL de manière simplifier afin que le développeur ne s'occupe que de la partie règle de métier.

1.1.3 Présentation de la différence entre ETL & ESB

L'ESB (Enterprise Service Bus) et l'ETL (Extract, Transform, Load) sont deux technologies positionnées sur le transport et la transformation de la donnée au sein du système d'information. Mais, historiquement, elles répondent à des objectifs différents. L'ESB permet des échanges de données fiables et sécurisés entre les différentes applications du SI tandis que l'ETL centralise et homogénéise les données de sources multiples vers une seule et même application de destination.

Concrètement, l'ESB est adaptée lorsqu'il s'agit de traiter une fréquence élevée de flux de données avec une volumétrie limitée. Plutôt orientée services, sa fonction est de transporter et décloisonner l'information pour la rendre accessible sur les différents outils métiers qui composent le SI.

Par ailleurs, l'ETL peut traiter un important volume de données de manière performante, mais sur un nombre d'échanges limité. Sa fonction est d'agréger toutes les informations pour traiter la donnée comme un ensemble standardisé. C'est une approche particulièrement adaptée pour les projets de Business Intelligence et de data warehousing.

1.2 Présentation d’un workspace

Les projet Talend sont stockés dans un espace de travail (workspace)

Chaque projet Talend contient un ensemble de Jobs (traduit sous forme de “classe”), ces jobs Talend sont stockés dans un projet Talend.

Ci-dessous, vous verrez l’arborescence d’un projet Talend :

- workspace
  - projet1
    - businessProcess (les Business Models)
    - code (les Routines)
    - configuration (les fichiers de configuration)
    - context (les contextes)
    - documentations (les documentations créés dans les jobs)
    - images (les images des tMap par exemple)
    - joblets (les joblets pour les version TIS)
    - metadata (les metadata)
    - process (les jobs)
    - sqlPatterns (les patterns sql)
    - temp (les fichiers temporaires)
  - projet2
    - businessProcess
    - code
    - …

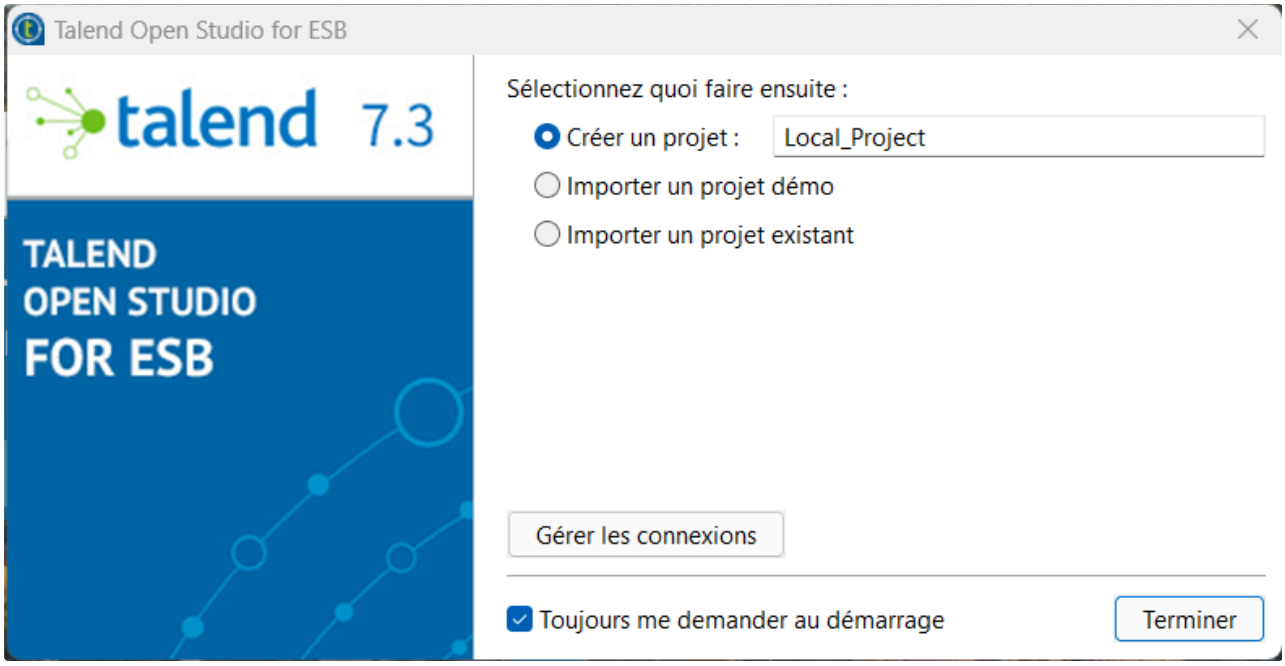
Avant de créer son premier workspace , il convient de lancer Talend Studio.

En supposant que les étapes précédentes ont été respectées, il convient de :

- Se rendre dans ce chemin : *C:/ESB_731/Studio*
- Faire un double clic gauche sur l’exécutable ***TOS_ESB-win-x86_64.exe*** afin de démarrer Talend Studio.

> Il est recommandé de créer un raccourci vers l’exécutable dans son Bureau afin d’éviter une navigation inutile.
>
> Rappel : CTRL+C sur l’exécutable puis CTRL+V sur le Bureau

À l’exécution vous devriez avoir la fenêtre suivante qui s’ouvre :



Par défaut, le WORKSPACE sera situé dans le chemin suivant :

- *C:/ESB_731/Studio/workspace*

Pour pouvoir changer d'endroit, il convient de :

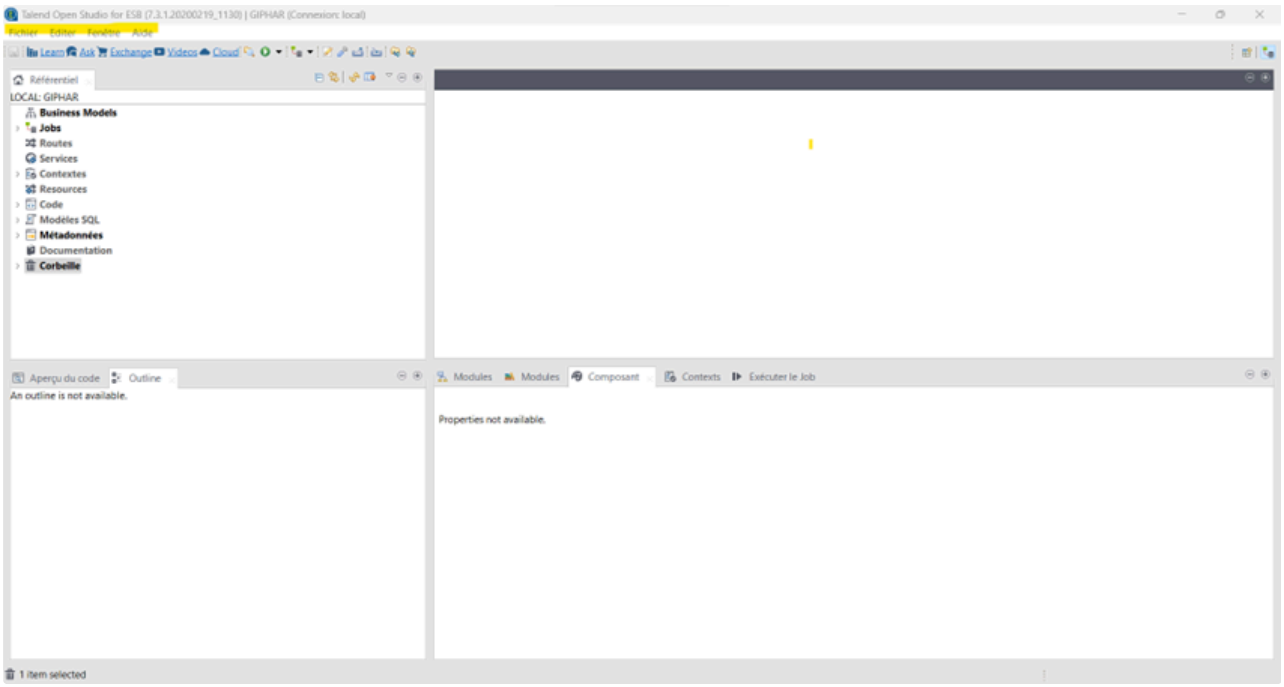
- Cliquer sur : Gérer les connexions
- Dans la partie Espace de travail : Mettre le chemin souhaité

1.3 Présentation du Studio

Rappel commande utile :

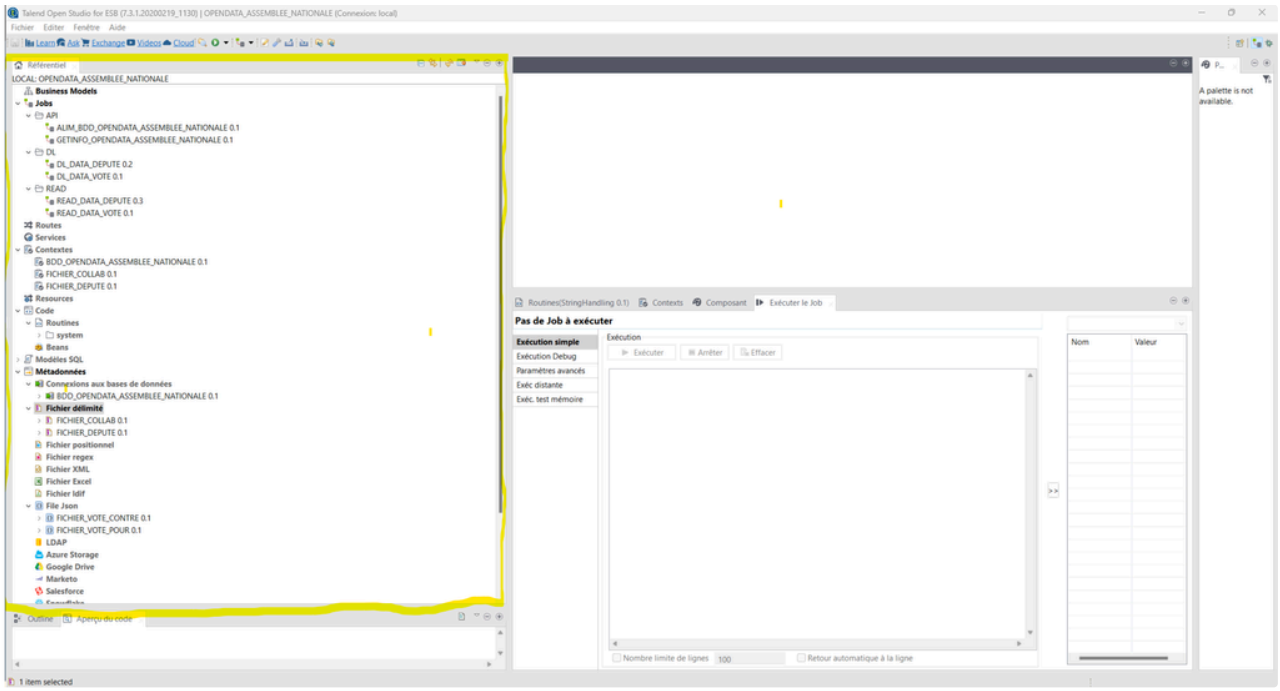
- CTRL + A : Tout sélectionner
- CTRL + C : Copier
- CTRL + V : Coller
- CTRL + Z : Revenir à l'état précédant une action
- CTRL + Y : Revenir à l'état succédant une action
- CTRL + S : Sauvegarder

1.3.1 Présentation de la Barre Principal



- FICHIER : Permets de changer de workspace et de modifier les propriétés du projet
- ÉDITER : Revenir en arrière, Copier, Coller, ect…
- FENÊTRE :
  - Modifier les éléments visuels du Studio
  - Modifier l'ensemble des paramètres pour les jobs du projet dans l'onglet Préférences

1.3.2 Présentation du Quadrant Nord Ouest



- BUSINESS MODELS : (Aucune idée de l'utilité ou non)
- JOBS : L'endroit où l'on peut retrouver les différents JOB, par JOB on entend un traitement de données.

- ROUTES : La principale différence entre les ROUTES et les JOB est que lorsque vous démarrez une route, elle écoute ou STREAM indéfiniment les entrées (fichier, message, etc.), et chaque fois qu'elles sont disponibles, elles sont traitées et envoyées à destination, jusqu'à ce que la route soit arrêtée.
  D'autre part, un JOB est un processus par lots ou BATCH qui est lancé à la demande pour gérer certaines entrées (fichiers, base de données, etc.) et se termine lorsque toutes les entrées sont traitées.

> Un point important à noter est que vous pouvez appeler des Jobs Talend à partir d'une Route Talend si vous le souhaitez.

- SERVICES : Un webservice est une fonction qui a pour rôle de mettre un disposition un service via internet. Le webservice est une interface entre deux applications, et leur permet tout comme l'API, de communiquer entre elles.
  Le webservice permet à des applications de communiquer entre elles même si elles fonctionnent avec des langages différents. Les webservices les plus connus sont de types SOAP, REST et HTTP.

> Les services de type REST peuvent se développer de la même manière qu'un job, seule la construction et le déploiement divergent.

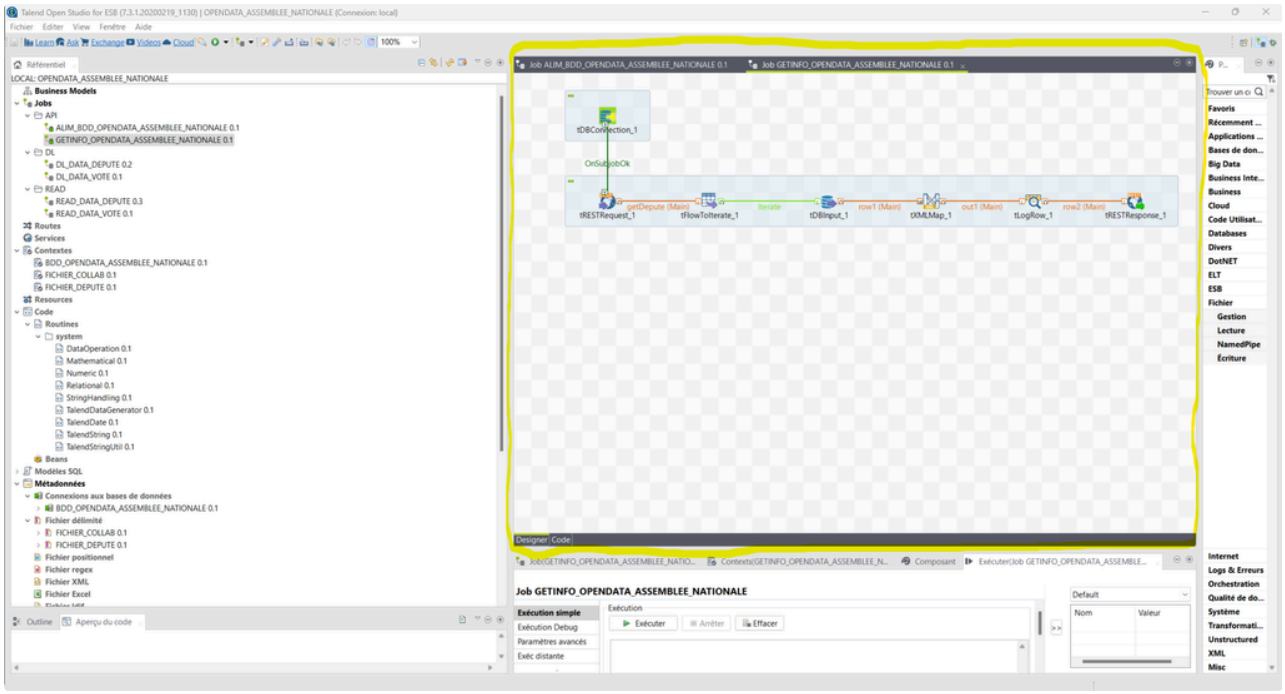
> Fonctionnement d'un WEBSERVICE
>
> **Étape 1** : Un utilisateur sur un ordinateur ou un mobile fait une demande. On l'appelle le client. Sa demande représente une requête qui est envoyée dans un langage spécifique : XML, HTTP ou encore JSON.
>
> **Étape 2** : La requête issue du client est envoyée sur un serveur distant via un protocole de type SOAP, REST ou HTTP.
>
> **Étape 3** : Le serveur va émettre une réponse qui aura le même format que celui du protocole de la demande.

- CONTEXTE : L'endroit où l'on peut retrouver les variables de contexte qui sont disponibles à l'exécution du JOB que l'on oppose aux variables globales qui sont disponibles dans le JOB.
  Ces variables peuvent être regroupées dans une catégorie qu'on appelle GROUPE et peuvent prendre leurs valeurs en fonction d'un ENVIRONNEMENT.
- RESSOURCE : (Aucune idée de l'utilité ou non)
- CODE : Permets de faire du code Java que l'on pourra réutiliser, on appelle ce code ROUTINES.

> Par défaut un certain de nombre de ROUTINES sont disponibles :
>
> - Mathematical : Permet de faire des opérations mathématiques sur des entiers ou des réels
> - TalendDate : Permet de faire des opérations sur des dates
> - StringHandling : Permet de faire des opérations sur des chaines de caractères

- MODEL SQL : (Aucune idée de l'utilité ou non)
- METADONNE : L'endroit où l'on stocke les **metadata** des diverses sources ou cibles de données possibles.
  On peut faire remarquer que ces **metadata** peuvent être paramétrées par des variables de contextes :
  - BDD
  - Fichier Délimité
  - Fichier JSON
  - …

1.3.3 Présentation du Quadrant Nord Est



L'endroit où nous allons créer les jobs qui prennent généralement des sources de données effectue des transformations en vue de les mettre dans une autre ou la même source de données.

Cette partie ne prend sens uniquement lorsqu'un JOB est ouvert.

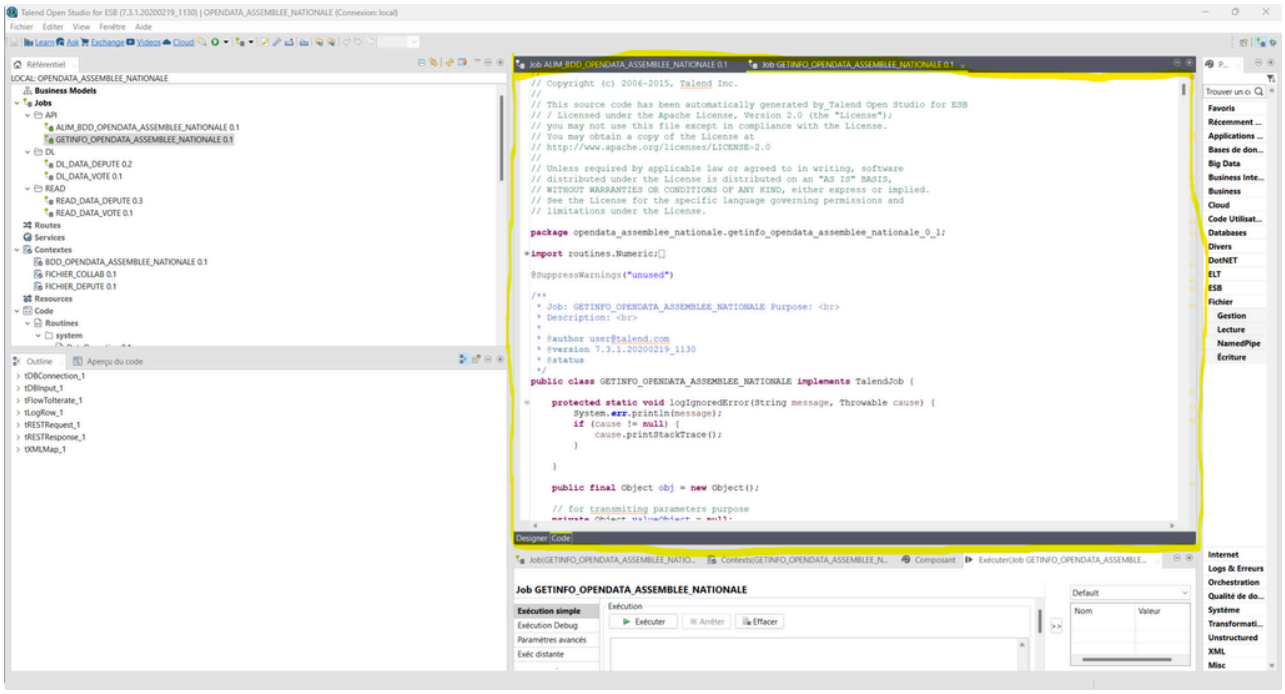
On importe des composants :

- Via la palette à droite
- En cliquant n'importe où dans la fenêtre puis en écrivant le nom du composant que l'on souhaite importer.

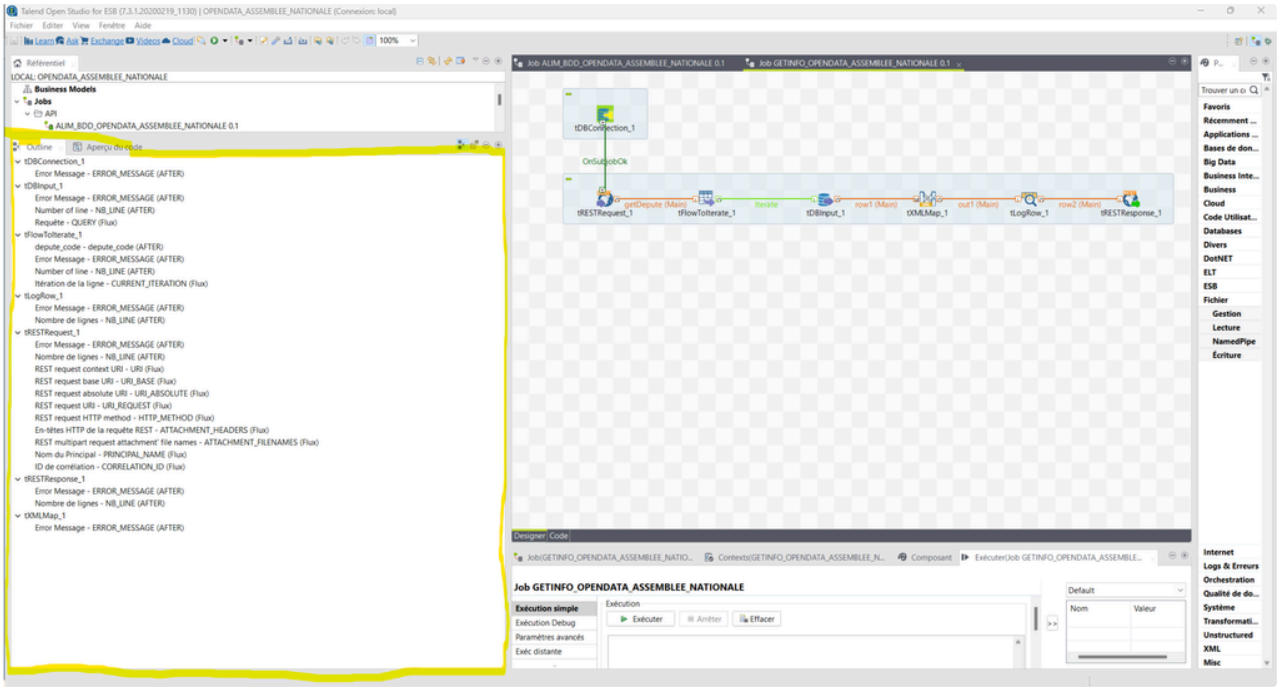
Par exemple : **tFileInputDelimited**

Dans l'onglet Code, on peut voir le code du job.
Celui-ci n'est là qu'à titre informatif et ne peut être modifié, il permet néanmoins de détecter d'éventuel erreur de compilation.

> Seule l'onglet Designer permet de développer.
>
> - Ne permet de détecter les erreurs qui peuvent se produire dans l'exécution du job
> - Permet de détecter les erreurs de compilation



1.3.4 Présentation du Quadrant Sud Ouest

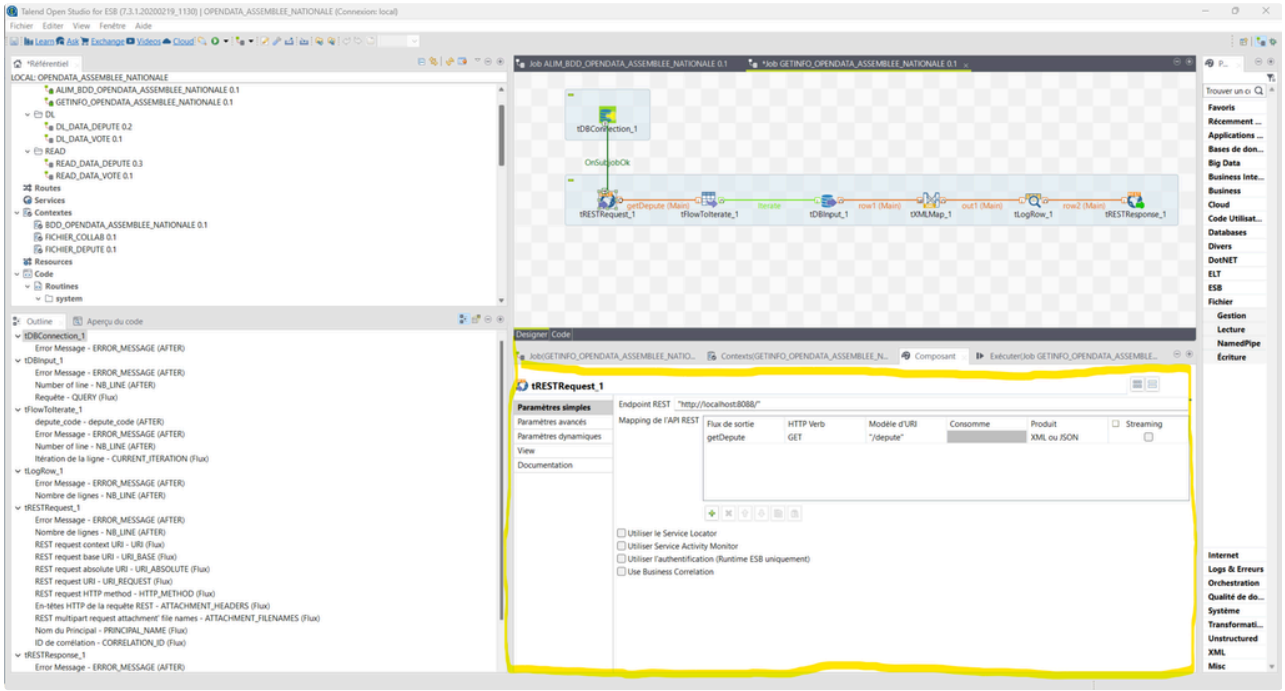


- APERÇU DU CODE : (Aucune idée de l'utilité ou non)
- OUTLINE : Ne prend sens que dans le cas où un JOB est ouvert.
  On retrouve ici l'ensemble des composants utilisés dans le JOB ouvert et les variables globales que ces composants génèrent.
  Ces variables peuvent être disponible avant ou pendant l'exécution du composant.

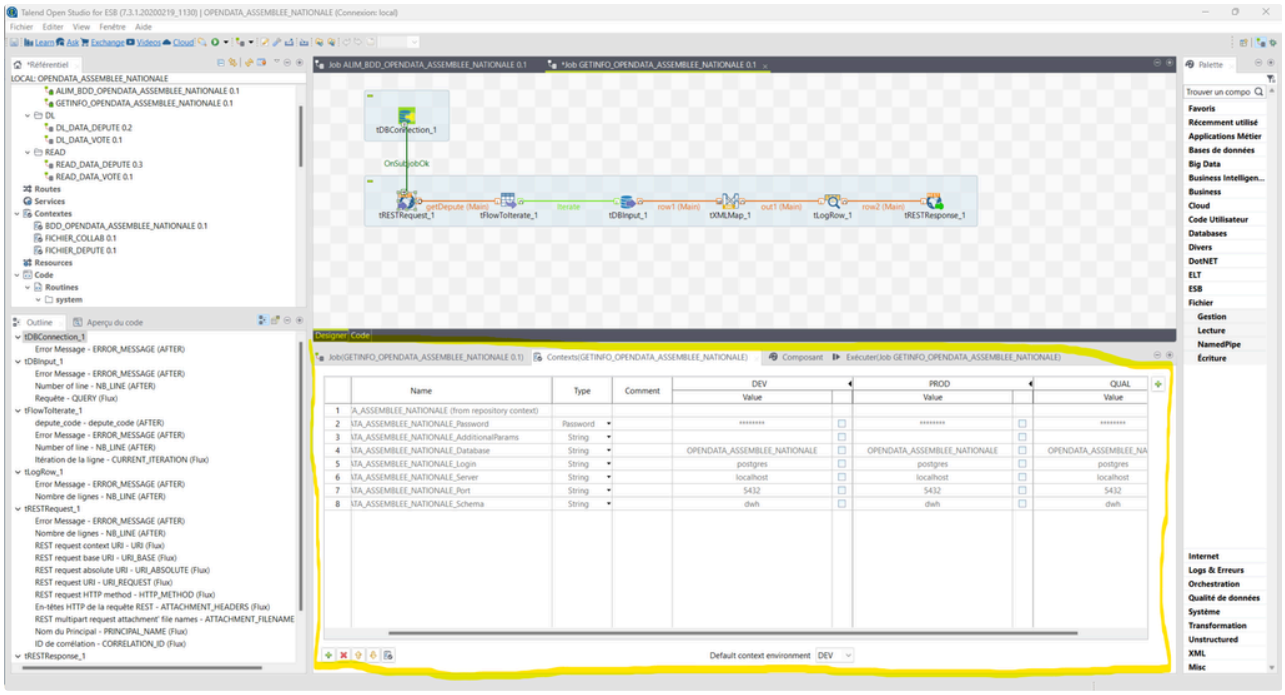
> Les variables globales définis par les composants, peuvent être appelés une fois définis

1.3.5 Présentation du Quadrant Sud Est

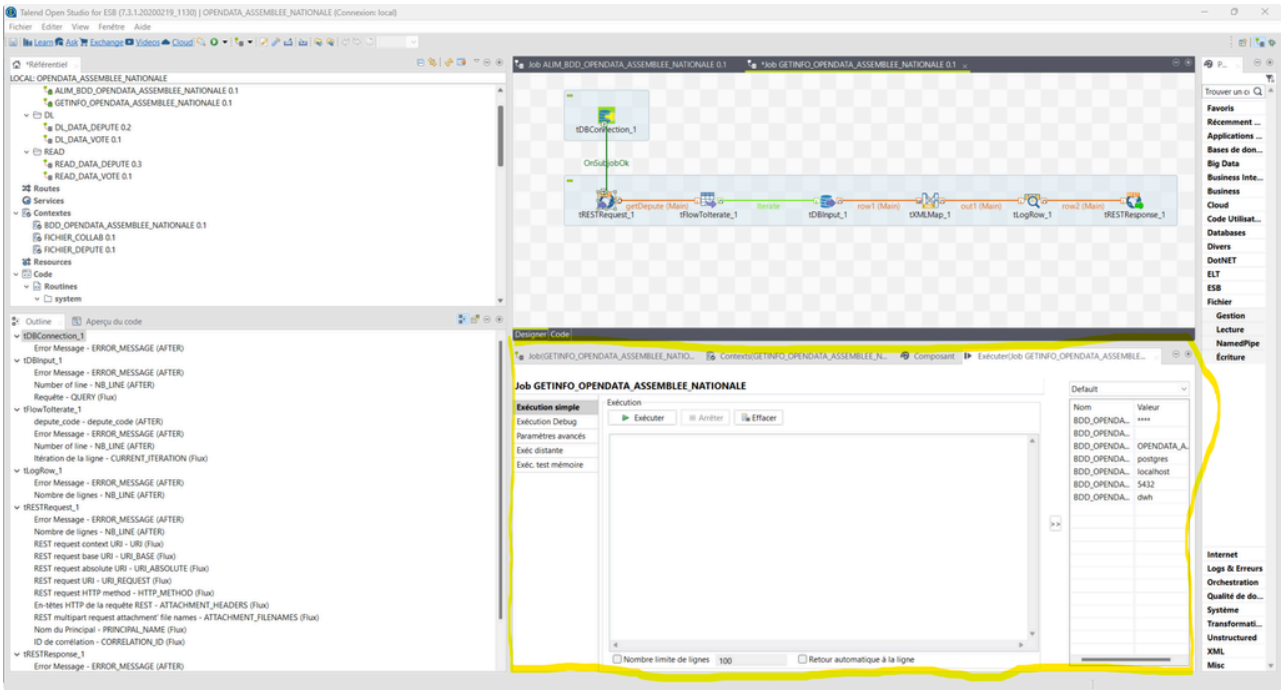
- COMPOSANT : Ne prend sens que dans le cas où un JOB est ouvert et qu'on a cliqué sur un composant.
  Dans cette fenêtre, nous allons paramétrer le composant afin que celui fonctionne selon nos besoins



- CONTEXT : Permet de visualiser et de gérer l'ensemble des variables de contexte utilisés dans le JOB



- EXÉCUTER LE JOB : Ne prend sens que dans le cas où un JOB est ouvert, il permet d'exécuter le traitement et d'afficher ce qui arrive dans la console. Une exécution en mode DEBUG est possible; une telle exécution permet d'afficher l'ensemble des variables utilisées par le programme et leurs valeurs à l'instant T. Il est également dans ce mode DEBUG de faire une pause dans le traitement. Deux types d'erreur peuvent se produire :
  - Les erreurs de **compilation** sont la conséquence d'un code mal écrit et se manifeste lors de la construction du job.
  - Les erreurs dans l'**exécution** sont la conséquence d'un code mal écrit ou mal définis et se manifeste lors de l'exécution du job.



1.4 Présentation d'un flux

Un job peut être vu comme un ensemble de composants liés les uns aux autres et se lit de gauche à droite puis du haut vers le bas ; à la manière de la lecture française.

1.4.1 Présentation des composants

Un COMPOSANT est un sous-ensemble d'un job qui effectue une opération définie. Par exemple : lire un fichier, filtrer des données ou encore extraire des données.

Un composant est composé d'un extrait de code Java généré automatiquement. Le but des composants est de gagner du temps en évitant le codage manuel pour effectuer les opérations de traitement de données courantes.

Talend propose une multitude de composants prêts à emploi regroupés un répertoire. Il est possible de développer des composants sur-mesure si Talend ne propose aucun composant répondant à votre besoin.

Ces composants nécessitent un paramétrage, le paramétrage peut être définis selon deux façons :

- **En dure** dans le sens où les valeurs associé aux paramètres du composant est définis par une valeur fixe écrit par le développeur
- **Avec variable** dans le sens où les valeurs associé aux paramètres du composant est définis par une valeur porté par une variable :
  - *contexte* : Disponible à l'exécution du job
  - *globale* : Définis dans le job

Les variables de contexte sont souvent utilisée pour porter les paramétrages de composant. Ces variables de contexte peuvent être regroupés dans un **GROUPE** et les valeurs portés par ces variables peuvent être définis en fonction d'environnement.

> Généralement, on définis au moins trois environnement.
>
> Prenons l'exemple d'un job visant à alimenter une BDD, on définis un groupe de contexte visant à porter les paramètres pour s'y connecter, on définit trois environnements :
>
> - **DEV** : BDD sur son poste permettant de faire son développement
> - **QUAL** : BDD sur un serveur dédié visant à simuler un traitement opérationnelle
> - **PROD** : BDD opérationnel

Certain composant peuvent être paramétrer via le concept de **metadata**, ce qui permet la réutilisation du composant avec le paramétrage associé.
On oppose ce concept de **metada** au fait de paramétrer le composant directement dans le job.

Certains paramétrages peut être fait de façon partiellement automatique ou à la main. C'est le cas notamment des composants de lecture de fichier, par exemple **tFileInputDelimited**

- **À la main** :
  - AVANTAGE : CONTRÔLE
  - DÉSAVANTAGE : VITESSE
- **Automatique** :
  - AVANTAGE : VITESSE
  - DÉSAVANTAGE : CONTRÔLE

> Il est recommandé de laisser Talend définir les **metadata** et de corriger le résultat si celui-ci ne convient pas

1.4.2 Présentation des liens

Les liens entre composants peuvent être de différentes natures :

- **Row** : Les lignes sont transférés de la sortie d'un composant à l'entrée d'un autres composant
  - *Main* : L'ensemble des lignes sont transférés
  - *Iterate* : Chaque ligne passe dans le composant de façon itérative
- **Trigger** : Le composant s'exécute en fonction d'une condition testé à la fin de l'exécution d'un autres composants
  - *On Component OK/ERROR* : Le composant s'exécute en fonction de si le composant précédent à réussi ou pas
  - *On Subjob OK/ERROR* : Le composant s'exécute en fonction de si le sous-job précédent càd une suite de composant lié en mode **Row** à réussi ou pas
  - *If* : Le composant s'exécute en fonction d'une condition qui retourne TRUE ou FALSE.

> On parle de sous-job lorsqu'une suite de composant est lié par des liens de type **Row**

Par convention, on lit les composants de gauche à droite lorsque le lien est de type **Row** et de haut en bas lorsque le lien est de type **If**

1.4.3 Présentation des variables

2. Installation des outils

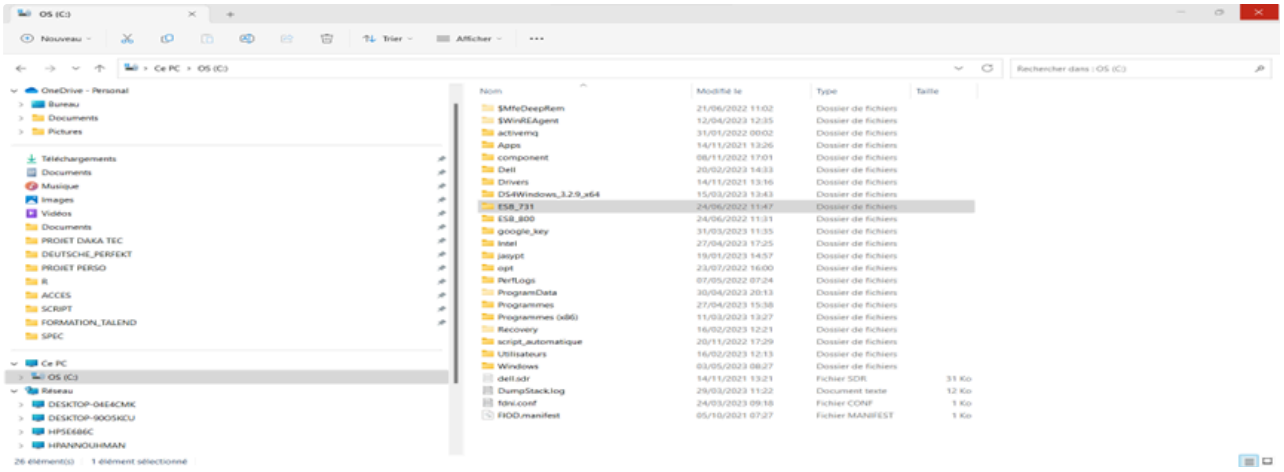
2.1 Installation de Talend

Étape 0 : Se rendre dans le dossier Google Drive suivant : google_drive

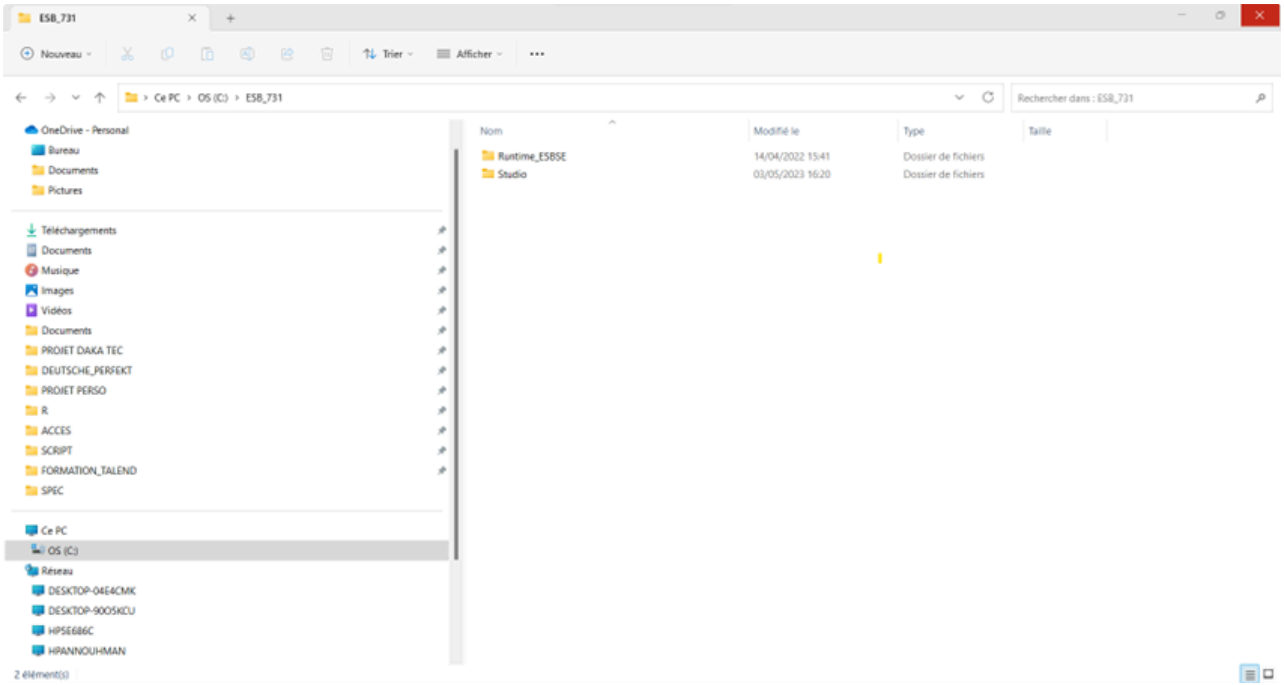
Étape 1 : Télécharger l'archive ***ESB_731.7z***

Étape 2 : Extraire l'archive téléchargée précédemment à la racine du disque *C:/*

À la fin de ces étapes, vous devriez obtenir le dossier suivant :



Également les sous-dossiers suivants :



2.2 Installation de Java

Talend peut être vu comme un générateur de code Java, ainsi il est donc nécessaire de disposer d'une version de machine java adapté à la version de Talend utilisé.

Étape 0 : Se rendre dans le dossier Google Drive suivant : google_drive

Étape 1 : Télécharger l'archive ***jdk-20.zip***

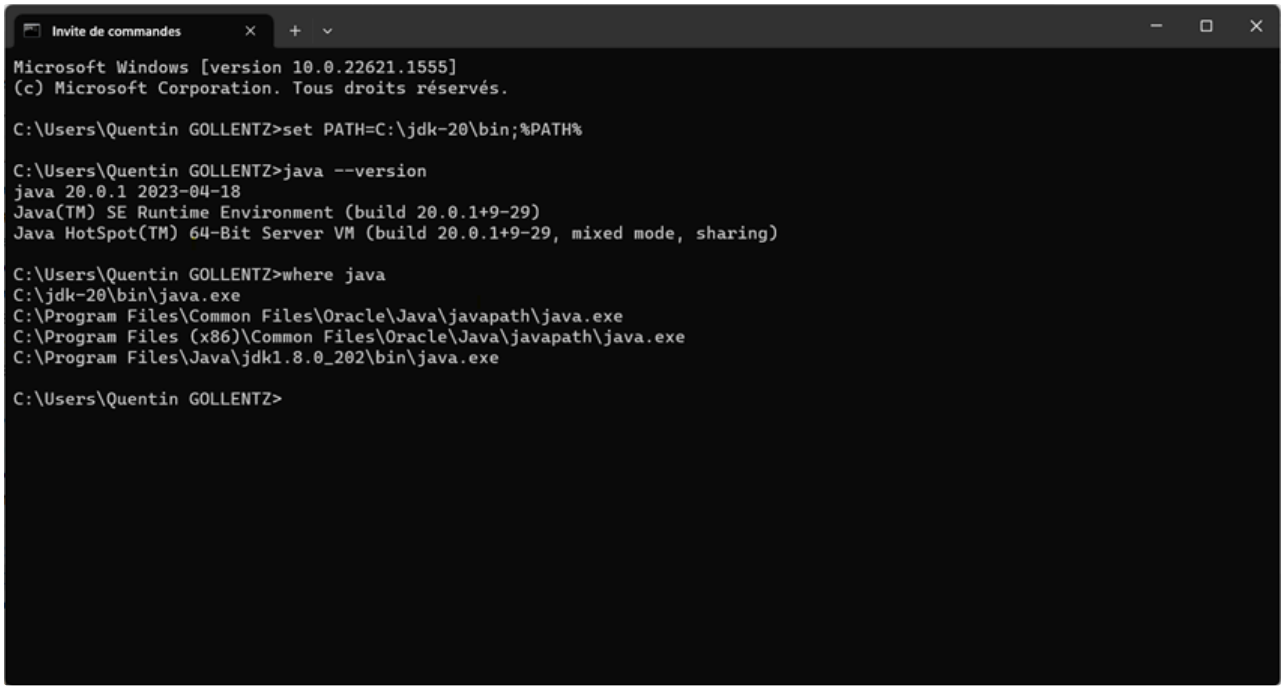
Étape 2 : Extraire l'archive téléchargée précédemment à la racine du disque *C:/*

Étape 3 : Lancer une invite de commande (Rappel : Écrire `cmd` dans une fenêtre)

Étape 4 : Lancer les commandes suivantes :

```
set PATH=C:\jdk-20\bin;%PATH%
java -version
where java
```

À la fin de ces étapes, vous devriez obtenir le résultat suivant :



Une autre solution est de spécifier le JAVA a utilisé par TALEND.

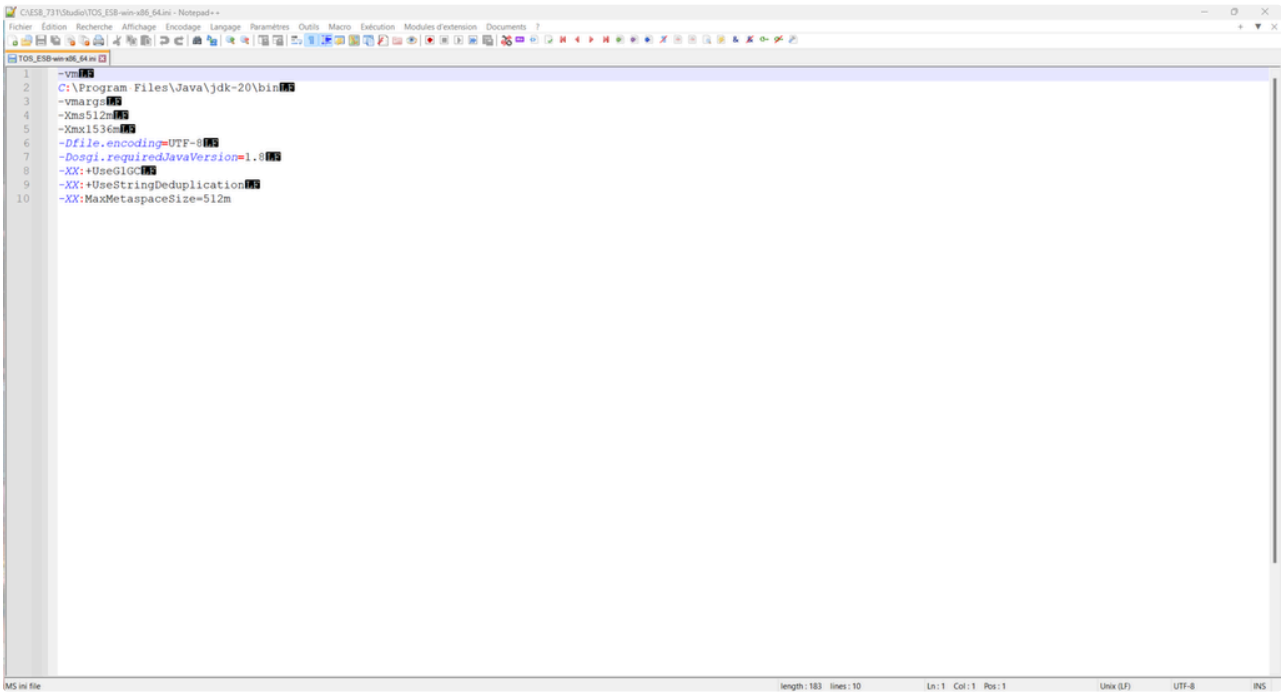
Étape 0 : Se rendre dans le dossier d'installation suivant suivant *C:/ESB_731* :

Étape 1 : Ouvrir le fichier `TOS_ESB-win-x86_64.ini`

Étape 2 : A début du fichier ajouter les deux lignes suivantes :

- `-vm`
  `C:\Program Files\Java\jdk-20\bin`

À la fin de ces étapes, vous devriez obtenir le résultat suivant :



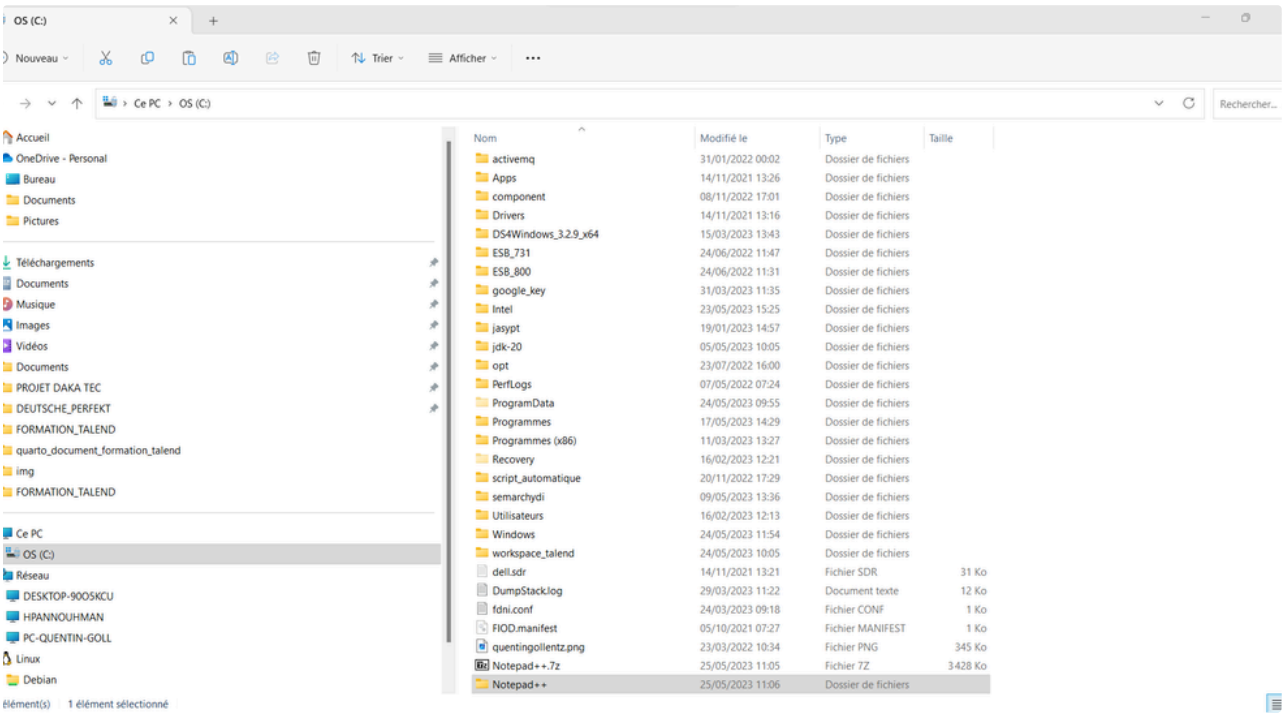
2.3 Installation de Notepad++

Étape 0 : Se rendre dans le dossier Google Drive suivant : google_drive

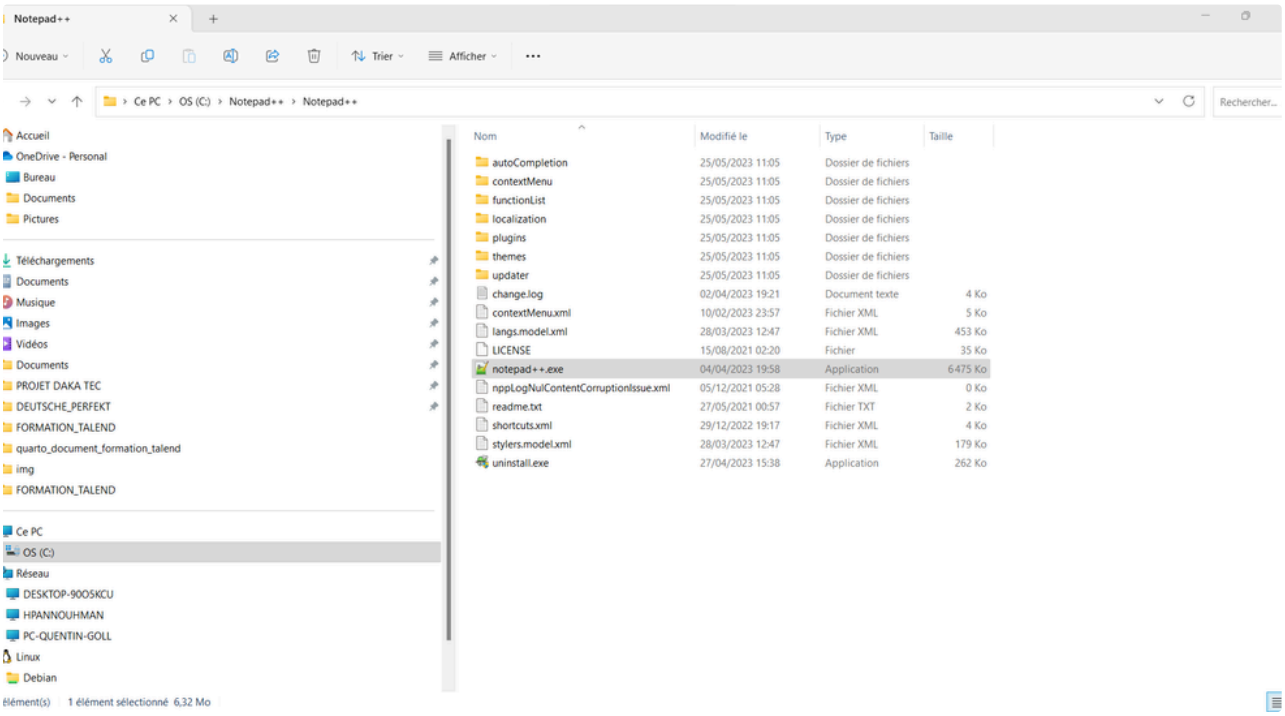
Étape 1 : Télécharger l'archive ***Notepad++.7z***

Étape 2 : Extraire l'archive téléchargée précédemment à la racine du disque *C:/*

À la fin de ces étapes, vous devriez obtenir le résultat suivant :



Également les sous-dossiers suivants :



On peut également télécharger des add-ons à Notepad :

- **JSON Viewer** : Pour mieux visualiser des fichiers JSON
- **XML Tools** : Pour mieux visualiser des fichiers XML
- **Compare** pour comparer deux fichiers

Pour pouvoir installer un add-on, il convient de :

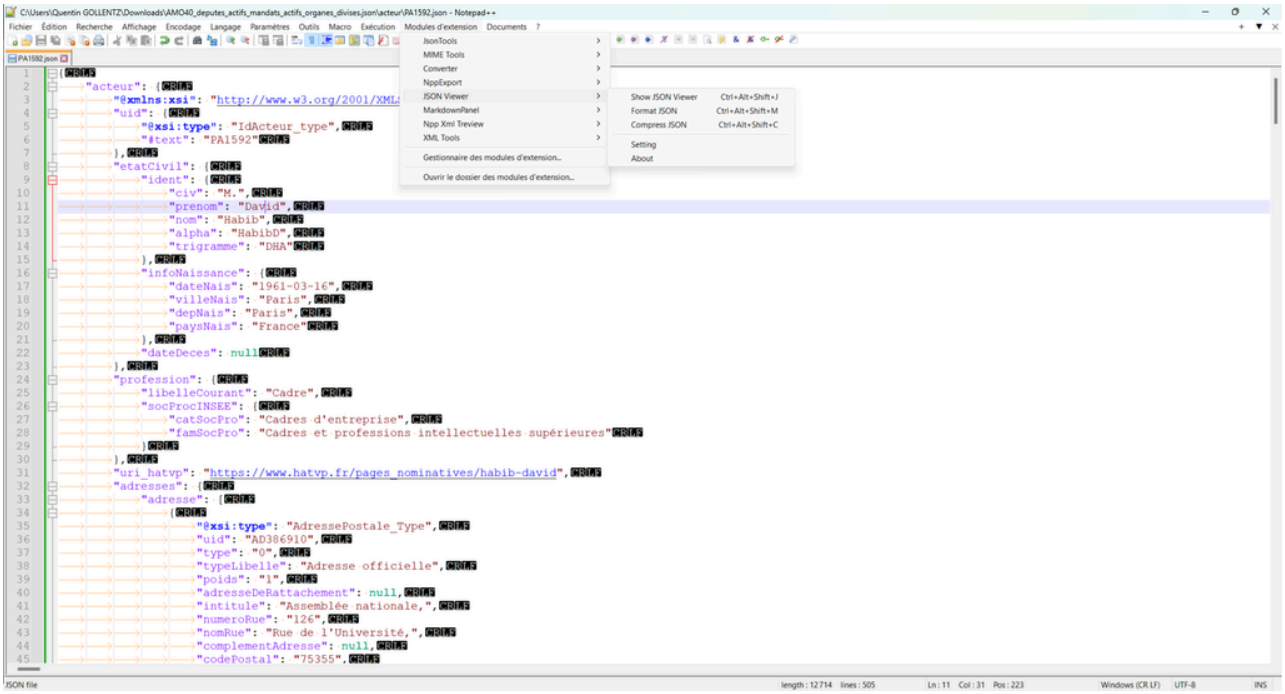
Étape 0 : Ouvrir Notepad

Étape 1 : Cliquer *Modules d'extension<Gestionnaires des modules d'extension*

Étape 2 : Rechercher **JSON Viewer** puis cocher (Faire de même avec **XML Tools** & **Compare**)

Étape 3 : Cliquer sur *Installer*

À la fin de ces étapes, vous devriez obtenir le résultat suivant :



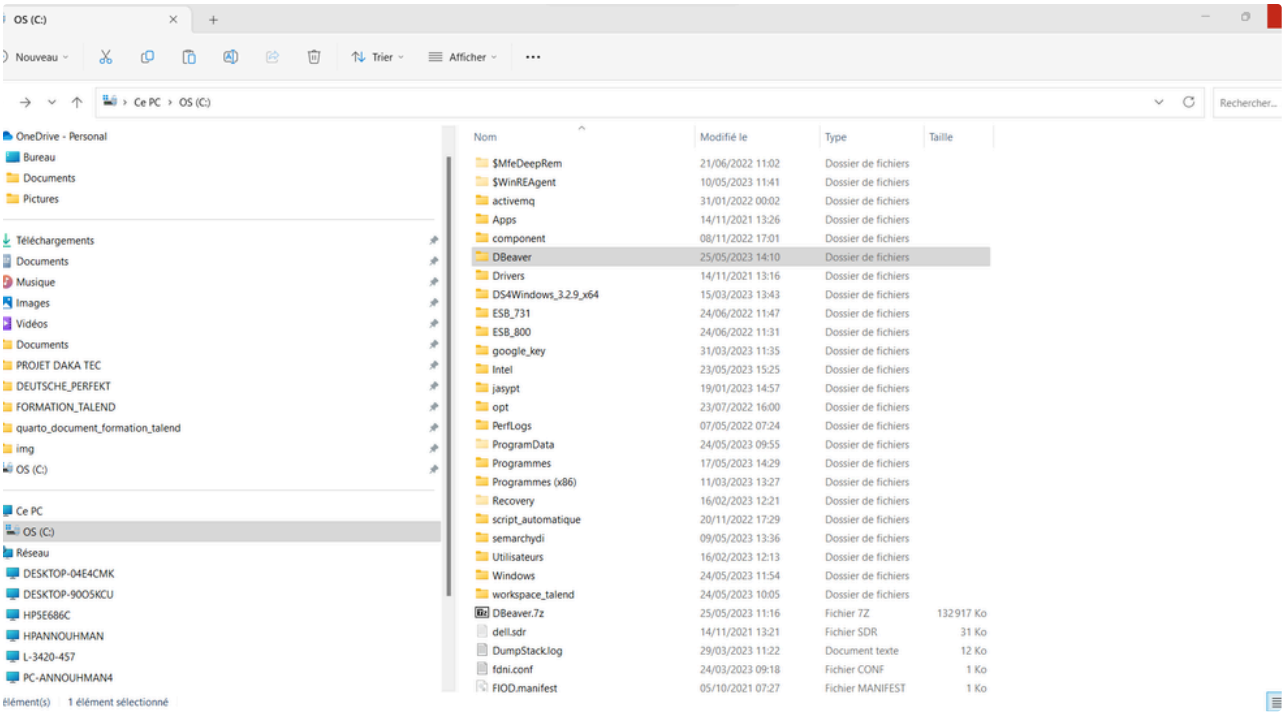
2.4 Installation de Dbeaver

Étape 0 : Se rendre dans le dossier Google Drive suivant : google_drive

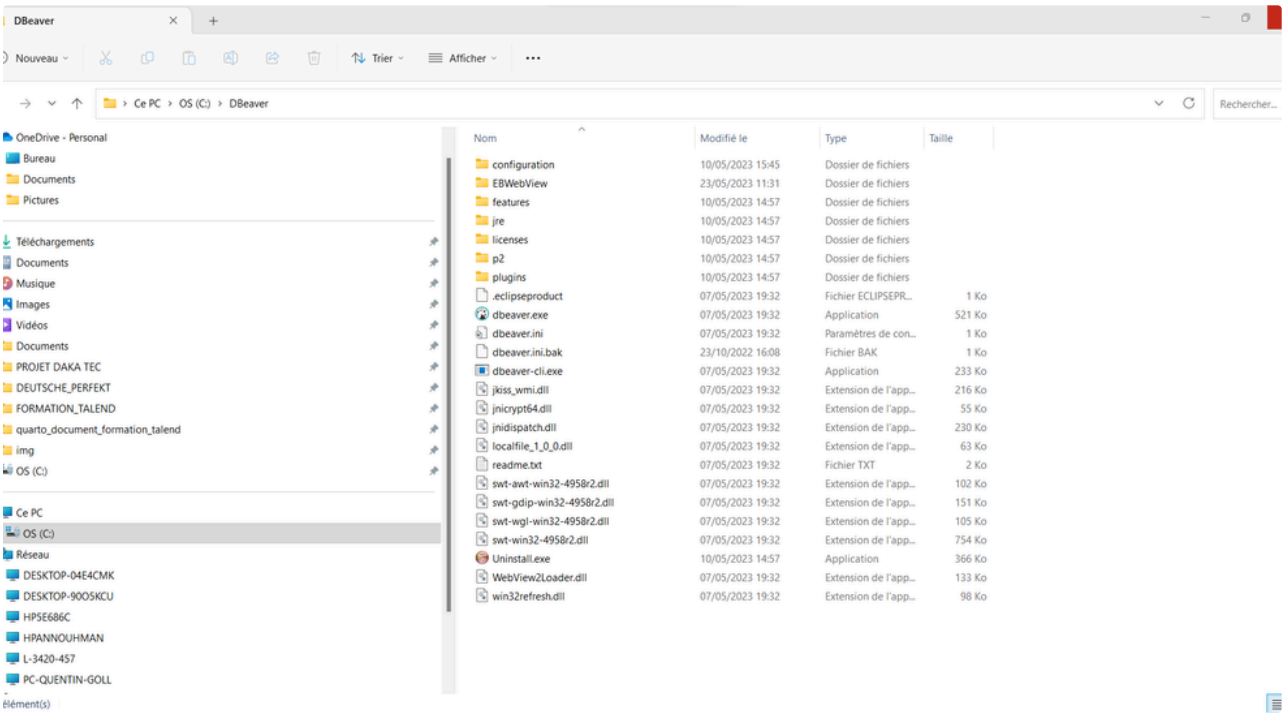
Étape 1 : Télécharger l'archive ***Dbeaver.7z***

Étape 2 : Extraire l'archive téléchargée précédemment à la racine du disque *C:/*

À la fin de ces étapes, vous devriez obtenir le résultat suivant :



Également les sous-dossiers suivants :



2.5 Installation de Postgresql

Expliquer les procédures serait redondant par rapport à la documentation qu'on peut trouver sur le web.
Vous pouvez suivre la procédure ici : postgresql

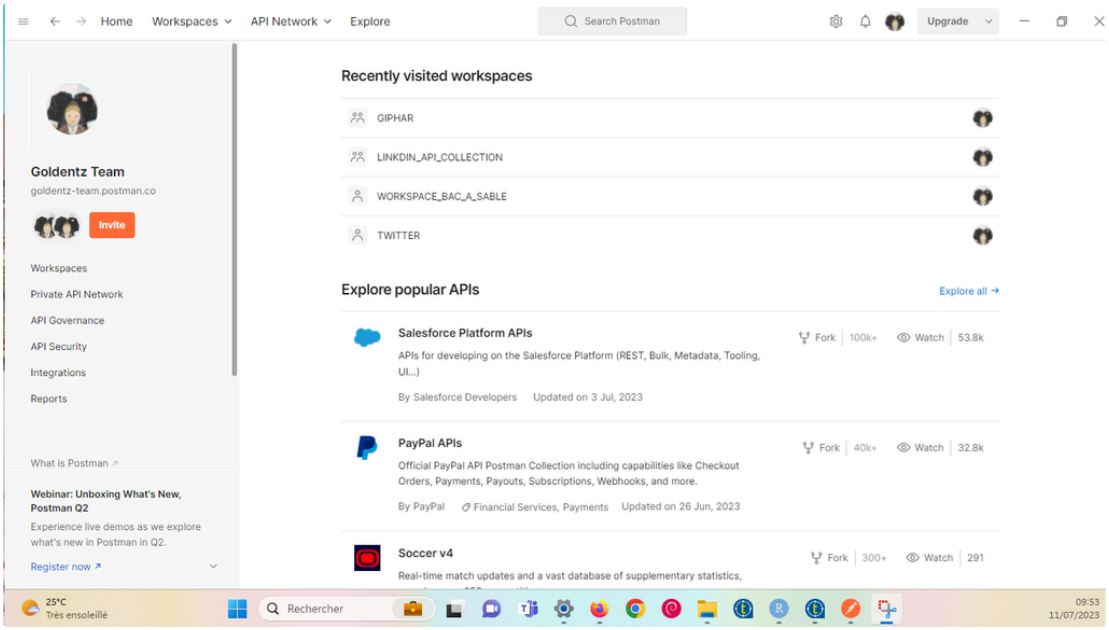
2.6 Installation de Postman

Étape 0 : Se rendre sur le site suivant : postman

Étape 1 : Télécharger l'archive ***Postman-win64-Setup.exe***

Étape 2 : Double-cliquer sur l'exécutable et laissez vous guider par l'assistant d'installation

À la fin de ces étapes, vous devriez obtenir le résultat suivant :

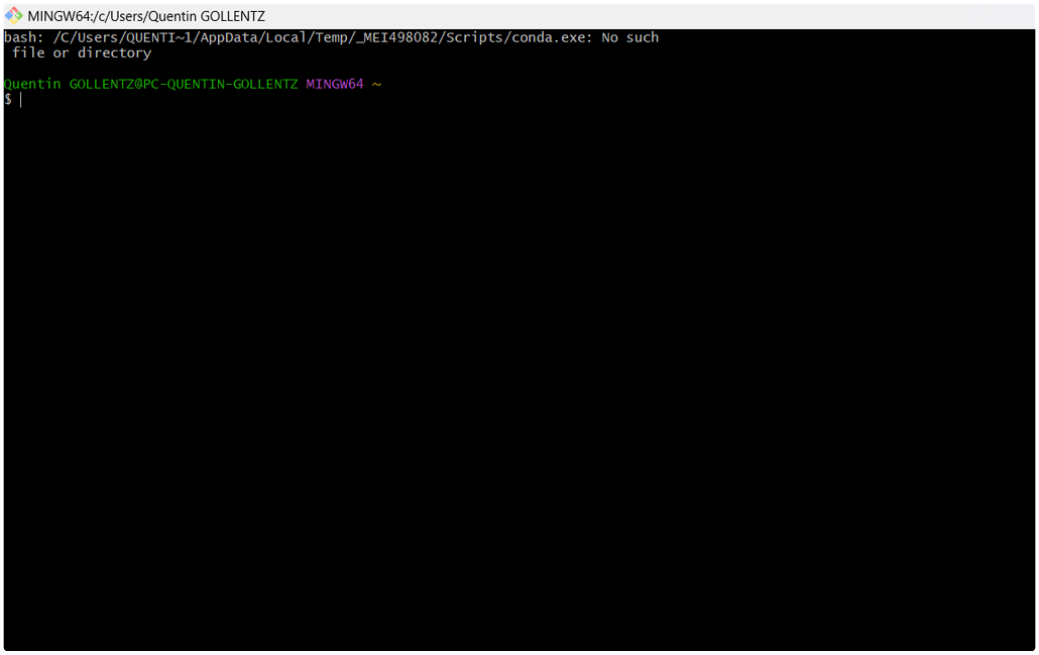


2.7 Installation de Git

Étape 0 : Se rendre sur le site suivant : git

Étape 1 : Installez le client GIT adapté à votre système d'exploitation, avec les options par défaut (cliquez sur Next à chaque fois).

À la fin de ces étapes, vous devriez obtenir le résultat suivant :



2. Norme et bonne pratique

Quand on cherche à évaluer ce qui est requis pour réaliser le meilleur code qui soit, les préceptes fondamentaux sont toujours de rigueur. Ils proviennent d'années d'expériences acquises sur nos erreurs et que nous avons améliorées avec succès. Ils représentent les concepts primordiaux qui créent les fondations sur lesquelles construire notre code et devraient être prises très au sérieux.

- La lisibilité: créer un code qui puisse être rapidement analysé et compris.
- Une écriture concise: vite et bien : créer rapidement, un code simple en un minimum de temps.
- La maintenabilité: réduire au stricte minimum la complexité afin d'avoir un impact tout aussi minime lors des évolutions.
- L'exactitude: créer un code qui réponde précisément au besoin.
- La réutilisabilité: créer des objets partageables et des tâches atomiques qui soient réutilisables.
- Le respect des règles: mettre en place une réelle discipline entre les équipes, les projets, les « repository », et le code. Autrement dit, imposer et respecter des règles de travail, de nommage, et d'écriture.
- La robustesse: créer un code qui plie mais ne casse pas. Autrement dit qui réagisse bien lorsque l'on s'écarte des conditions normales d'utilisation.
- L'extensibilité: créer des modules élastiques qui puissent s'adapter à la demande.
- La cohérence: créer des choses basiques avant tout.
- L'efficience: réaliser des flux de données et composants optimaux.
- Le cloisonnement: créer des modules atomiques, ciblés qui répondent à un seul et unique besoin. Évitez les couteaux suisses !
- L'optimisation: réaliser le plus de fonctionnalités possibles avec le moins de code possible.
- La performance: réaliser des modules efficaces qui fournissent les débits les plus rapides.

Parvenir à un véritable équilibre entre ces préceptes est la clef : en particulier en ce qui concerne les 3 premiers, car ils sont en parfaite contradiction les uns avec les autres. Vous en obtenez souvent 2 au sacrifice du troisième. Tâchez alors de les ordonner par ordre d'importance, si vous le pouvez !

Notre conseil pour bien pratiquer Talend : rester le plus simple possible !

- Nommer les composants, mettre des titres et des notes
- Attention à la lisibilité
- Éviter la barre de défilement horizontale, aligner les sous-jobs
- Au-delà d'une trentaine de composants, se poser la question de la division en plusieurs jobs (père-fils?)
- Attention à l'organisation et aux noms des dossiers contenant vos jobs

2.1 Variable

Il convient de variabiliser :

- Toute metada dont la valeur est susceptible de changer d'un environnement à un autre. On définis alors autant de groupe qu'il existe d'environnement. On utilisera généralement des **variables de contexte**.
- Toute metada qui est amené à être réutilisé dans différents jobs. On utilisera généralement des **variables de contexte**
- Toute metada qui est amené à être utilisé plusieurs fois au sein d'un même job. On utilisera généralement des **variables globales**.

2.1.1 Variable de contexte

Une variable de contexte peut avoir différentes valeurs selon le contexte (DEV, QUAL, PRD). Elles permettent d'exécuter le même job dans différents contextes / environnements.

Elles peuvent être utilisées dans tout le projet en activant "Transmettre tout le contexte" dans les **tRunJob**. Il est possible de revaloriser la variable dans un job.

Celles-ci sont appelés par la syntaxe suivante

```
context.NOM_VARIABLE
```

2.1.2 Variable globale

Elles sont liées à un composant et générées automatiquement en fonction du type de composant. Elles ne sont accessibles que dans le job courant. (scope = portée). Elles permettent d'accéder à l'état du composant lié :

- Nom du fichier
- Numéro de l'itération courante
- Message d'erreur

Elle permette de savoir ce qu'il se passe, d'identifier les problème afin d'avoir une meilleur maintenance et suivie, mais également de créer du code dynamique

On peut faire remarquer que celles-ci peuvent être définis par le développeur avec la syntaxe suivante

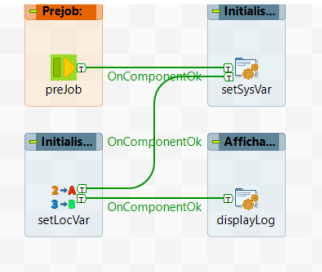
```
(Type)globalMap.put("nom de variable", "la valeur");
```

Celles-ci sont appelés par la syntaxe suivante

```
(String)globalMap.get("nom de variable");
```

2.2 Gestion du monitoring

2.2.1 Gestion du début du traitement



Ce sous-job n'est pas directement lié au traitement du job lui même, mais permet de définir un certain nombre de paramètre lié à l'exécution du job lui-même.

Généralement, on réalise les actions suivantes :

- On spécifie une locale
- On définit la date d'exécution de la route : **P_VAR_JOB_STA_DAT** avec pour valeur `TalendDate.getCurrentDate()`
- On affiche un log de début de traitement

```
System.out.println("---------------");
System.out.println("DEBUT EXECUTION");
System.out.println("---------------");
System.out.println("Nom du projet: "+projectName);
System.out.println("Nom du job: "+jobName);
System.out.println("Date Debut: "+TalendDate.formatDate("dd/MM/yyyy HH:mm:ss.SSS", (Date)globalMap.get("P_VAR_JOB
```

> On peut également effectuer dans ce cadre un chargement du contexte via un fichier ce qui permet de simplifier le déploiement sur différents environnements en gestion.

> Ce fichier de contexte peut être chiffré et déchiffré par Talend ce qui permet notamment de ne pas laisser de façon visible les paramètres d'accès à des bases de données par exemple.

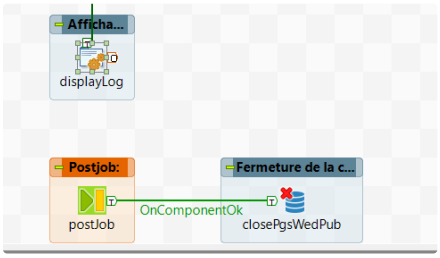
2.2.2 Gestion de la fin du traitement

On distingue deux étapes pour la fin de job :

- La fin du traitement (fonctionnelle) qui ne produit que lorsque le traitement n'a connu aucune erreur. Dans cette Étape , on se contente d'afficher un log de fin de traitement

```
System.out.println("-------------");
System.out.println("FIN EXECUTION");
System.out.println("-------------");
System.out.println("Nom du projet: "+projectName);
System.out.println("Nom du job: "+jobName);
System.out.println("Date Debut: "+TalendDate.formatDate("dd/MM/yyyy HH:mm:ss.SSS", (Date)globalMap.get("P_VAR_JOB
System.out.println("Date Fin: "+TalendDate.formatDate("dd/MM/yyyy HH:mm:ss.SSS", TalendDate.getCurrentDate()));
System.out.println("Statut: OK");
```

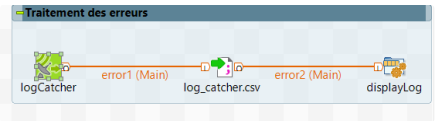
- La fin du job (technique) qui se produit dans tout les cas (même en cas d'erreur). On utilise généralement un composant **tPreJob** pour clôturer les connexion à des bases de données.



2.2.3 Gestion des erreurs du traitement

Dans le cas d'erreur dans l'exécution du traitement, il est nécessaire d'obtenir les informations liés à l'erreur afin de pouvoir corriger le rapidement.

Pour cela, nous utilisons un sous-job qui utilise notamment un composant **tLogCatcher** qui permet de récupérer les logs de traitements pour les écrire ensuite dans un fichier, base de données, ect ..



2.2.4 Gestion des logs du traitement

Nous souhaitons également obtenir des informations sur le traitement lui-même, pour cela nous pouvons utiliser du code Java dans un composant **tJava** pour afficher les informations dont nous avons besoin.

Exemple :

```
String groupe = (String)globalMap.get("groupe");

if (Relational.ISNULL(groupe)) {
    globalMap.put("filtreGroupe","");
    globalMap.put("groupe","tous");
    System.out.println("----------");
    System.out.println("Aucune groupe parametree: tous les députés sont pris en compte");
```

```
    System.out.println("----------");
} else {
    globalMap.put("filtreGroupe","AND groupe_type_cd = '"+groupe+"'");
    System.out.println("----------");
    System.out.println("Groupe en cours de traitement: "+groupe);
    System.out.println("----------");
}
```

2.3 Règle de nommage et esthétique

Quelques règles spécifiques à suivre :

- Ces caractères spéciaux sont interdits : la barre oblique (/), l'esperluette (&), et la virgule (,).
- Éviter d'utiliser des espaces blancs ou des caractères spéciaux.
- Remplacer les espaces vides par des tirets bas.

On recommande la lecture de haut en bas et de de gauche vers la droite.

Ainsi on aligne de gauche à droite les composants au sein d'un sous-job et de haut en bas les sous-jobs entre eux.

2.3.1 Règle de nommage des composants

On utilise le camelCase pour décrire l'action réalisé par le composant en question.

Exemple : **displayLog** pour un composant **tJava** permettant d'afficher des logs

2.3.2 Règle de nommage des liens

On utilise les minuscules reprenant le nom du composant générant l'output en ajoutant `_output`

Exemple : Pour le lien de sortie de type *Main* d'un composant **tPostreSQLInput** nommé **getData**, on peut nommer le lien de sortie `getdata_output`

2.3.3 Règle de nommage des sous-jobs

On écrit une phrase la plus courte permettant de décrire l'action réalisé

Exemple : Un sous-job permettant de récupérer des données peut être résumé par la phrase :

Récupération des données

2.3.4 Règle de nommage des jobs

On utilise les majuscules en essayant de décrire l'action du job

Exemple : **DL_DATA** pour un job qui vise à télécharger des fichiers

Il convient également d'apporter une description dans les propriétés du job.

Exemple :

```
##########
DATE DE CREATION : {dd/MM/yyyy}
DATE DE MODIFICATION :
VERSION : 0.1
AUTEUR : {nom_prenom}
DESCRIPTION : CREATION DU JOB
##########
```

A chaque évolution, il convient d'incrémenter la version du flux afin de garder un historique des versions

3. Création d'un système d'information

3.1 Création d'un projet

> Nous allons dans cette partie créée notre premier et unique projet.

Étape 0 : Créer un dossier : *C:/workspace_talend*

Étape 1 : Démarrer Talend

Étape 2 : Cliquer sur "Gérer les connexions" et choisir le dossier *C:/workspace_talend* créé en Étape 0

Étape 3 : Attendre que Talend redémarre

Étape 4 : Créer un projet nommé : **OPENDATA_ASSEMBLEE_NATIONALE**

3.2 Obtention des données

3.2.1 Création du job DL_DATA

> L'objectif ici est de télécharger les fichiers source que nous allons utilisé par la suite.

> - **tFileFetch** : Ce composant récupère un fichier via un protocole donné (HTTP, HTTPS, FTP ou SMB).

Étape 0 : Créer un job avec :

- pour titre : **DL_DATA**
- pour objectif : Téléchargement d'un fichier de type .**csv** contenant les informations des députés
- pour description le bloc suivant en remplaçant les {} par vos informations :

```
##########
DATE DE CREATION : {dd/MM/yyyy}
DATE DE MODIFICATION :
VERSION : 0.1
AUTEUR : {nom_prenom}
DESCRIPTION : CREATION DU JOB
##########
```

Étape 1 : Créer un dossier dans lequel nous allons mettre des fichiers brut non généré par le job qu'il prend en entrée

- *C:/workspace_talend/source/*

Étape 2 : Créer un dossier dans lequel nous allons mettre des fichiers généré par le job qu'il génère en sortie

- *C:/workspace_talend/cible/*

Étape 3 :

- Importer le job TEMPLATE_JOB.zip
- Importer le groupe de contexte **P_VAR** dans le job **DL_DATA**

Étape 4 : Créer un groupe de contexte **DIR** avec trois groupes (DEV, QUAL,PROD)

	DEV	QUAL	PRD
DIR_RACINE	`C:/workspace_talend/`		

Étape 5 : Copier l'intégralité de **TEMPLATE_JOB** dans le job **DL_DATA**

Étape 6 : Pour effectuer les téléchargements, nous allons utilisé le composant **tFileFetch** qui peut être trouvé soit dans la palette à droite, soit en tapant directement **tFileFetch** n'importe où dans le QUADRANT NORD EST .

- Soit le paramétrage suivant pour députés :

```
Nommage du composant : getDepute
Nommage du sous-job: Obtention des données des députés
##########
URL : "https://www.data.gouv.fr/fr/datasets/r/092bd7bb-1543-405b-b53c-932ebb49bb8e"
MODE : HTTPS
METHODE : GET
NOM_FICHIER : "depute.csv"
CIBLE : context.DIR_RACINE + "/source/depute/"
ARRET EN CAS D'ERREUR
##########
```

- Soit le paramétrage suivant pour collaborateurs :

```
Nommage du composant : getCollaborateur
Nommage du sous-job: Obtention des données des collaborateurs des députés
##########
URL : "https://data.assemblee-nationale.fr/static/openData/repository/16/amo/collaborateurs_csv_opendata/liste_c
MODE : HTTP
METHODE : GET
NOM_FICHIER : "collaborateur.csv"
CIBLE : context.DIR_RACINE + "/source/collaborateur/"
ARRET EN CAS D'ERREUR
##########
```

- Soit le paramétrage suivant pour votes:

```
Nommage du composant : getVote
Nommage du sous-job: Obtention des données de vote
##########
URL : "http://data.assemblee-nationale.fr/static/openData/repository/16/loi/scrutins/Scrutins.json.zip"
MODE : HTTP
METHODE : GET
NOM_FICHIER : "vote.zip"
CIBLE : context.DIR_RACINE + "/source/vote/"
ARRET EN CAS D'ERREUR
##########
```

Étape 7 : Relier chaque composant **tFileFetch** par un lien de type *OnSubjobOk*

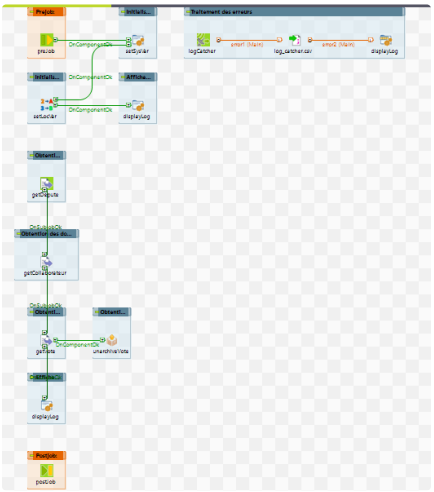
Étape 8 : Extraire l'archive **vote.zip** avec un composant **tFileUnarchive** avec pour paramétrage suivant :

```
Nommage du composant : unarchiveVote
Nommage du sous-job: Obtention des données de vote
##########
FICHIER D'ARCHIVE : context.DIR_RACINE + "/source/vote/vote.zip"
REPERTOIRE D'EXTRACTION : context.DIR_RACINE + "/source/vote/"
##########
```

Étape 9 : Lier l'exécution du composant nouvellement créé avec le composant **tFileFetch(getVote)** , en cliquant droit sur le premier composant, puis choisir *OnComponentOk*

Étape 10 : Relier le dernier **tFileFetch(getVote)** au dernier **tJava(displayLog)** issue du **TEMPLATE_JOB**

Votre job devrait ressembler à ça :



3.2.2 Exécution du job

> L'objectif ici est de vérifier la bonne exécution du job.
> Ici nous souhaitons simplement télécharger des fichiers, il convient donc de s'assurer que les fichiers sont bien présents dans le dossier spécifié.

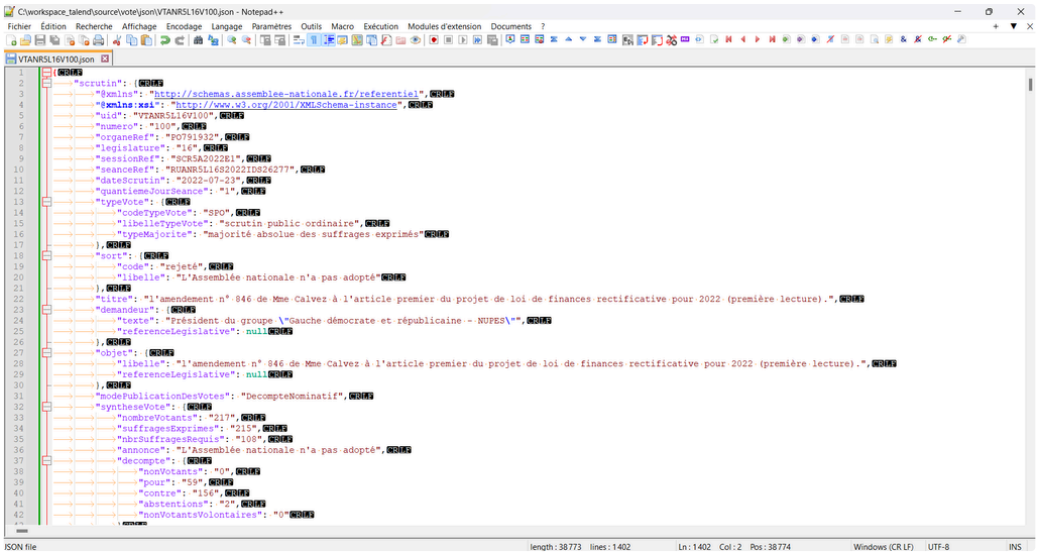
Étape 0 : Se mettre dans l'onglet "EXÉCUTER" dans le QUADRANT SUD EST .

Étape 1 : Cliquer EXÉCUTER en utilisant l'environnement **DEV** et observer qu'aucune erreur de compilation n'est générée.

Étape 2 :

- Vérifier la présence du fichier *C:/workspace_talend/source/depute/**depute.csv***
- Vérifier la présence du fichier *C:/workspace_talend/source/collaborateur/**collaborateur.csv***
- Vérifier la présence du fichier *C:/workspace_talend/source/vote/**vote.zip***
- Vérifier la présence de fichiers **.json** dans *C:/workspace_talend/source/vote/json*

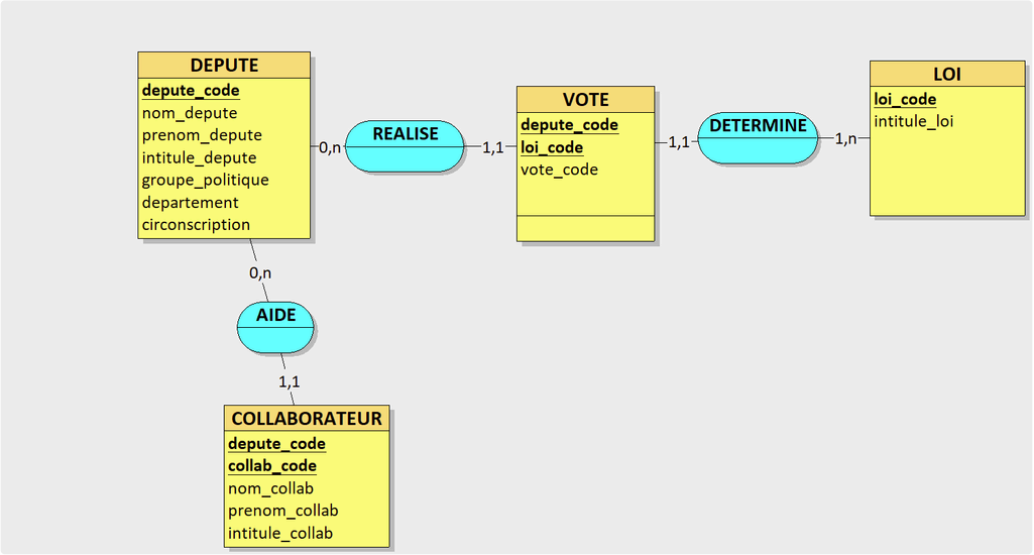
> Nous vous invitons à ouvrir quelques fichiers avec **Notepad++** notamment les fichier **.json**



3.3 Alimentation de la base de données

3.3.1 Création de la BDD

Nous souhaitons obtenir le MPD suivant :



BDD : OPENDATA_ASSEMBLEE_NATIONALE

Étape 0 : Ouvrir **PgAdmin** et construire une database avec pour nom `OPENDATA_ASSEMBLEE_NATIONALE` et un schéma avec pour nom `dwh` :

```
DROP DATABASE IF EXISTS "OPENDATA_ASSEMBLEE_NATIONALE";

CREATE DATABASE "OPENDATA_ASSEMBLEE_NATIONALE"
  WITH
  OWNER = postgres
  ENCODING = 'UTF8'
  LC_COLLATE = 'French_France.1252'
  LC_CTYPE = 'French_France.1252'
  TABLESPACE = pg_default
  CONNECTION LIMIT = -1;
```

```
11
12 COMMENT ON DATABASE "OPENDATA_ASSEMBLEE_NATIONALE"
13   IS 'Base de données contenant les informations issus de l'OpenData de l'Assemblée Nationale';
14
15 DROP SCHEMA IF EXISTS dwh ;
16
17 CREATE SCHEMA IF NOT EXISTS dwh
18   AUTHORIZATION postgres;
```

Étape 1 : Créer les diverses tables

- Table **DEPUTE** :

```
1 CREATE TABLE dwh."DEPUTE" (
2   depute_code varchar(20) NOT NULL,
3   nom_depute varchar(250) NULL,
4   prenom_depute varchar(250) NULL,
5   intitule_depute varchar(250) NULL,
6   groupe_politique varchar(250) NULL,
7   departement varchar(250) NULL,
8   circonscription integer NULL
9 );
```

- Table **COLLAB** :

```
1 CREATE TABLE dwh."COLLAB" (
2  collab_code integer NOT NULL,
3  depute_code varchar(20) NOT NULL,
4  nom_collab varchar(250) NULL,
5  prenom_collab varchar(250) NULL,
6  intitule_collab varchar(250) NULL
7 );
```

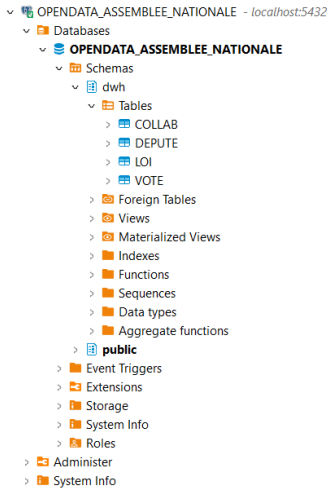
- Table **VOTE** :

```
1 CREATE TABLE dwh."VOTE" (
2   depute_code varchar(20) NOT NULL,
3   loi_code varchar(250) NOT NULL,
4   vote_code integer NULL
5 );
```

- Table **LOI** :

```
1 CREATE TABLE dwh."LOI" (
2   loi_code varchar(250) NOT NULL,
3   intitule_loi varchar(1000) NULL
4 );
```

Vous devriez avoir le résultat suivant avec **DBeaver** :



3.3.2 Création du job ALIM_BDD

L'objectif est d'alimenter la BDD

- **tPostreSQLConnection**: Ce composant créer une connexion à une BDD de type PostgreSQL
- **tPostreSQLClose** : Ce composant ferme une connexion à une BDD de type PostgreSQL

Étape 0 : Créer un job avec :

- pour titre : **ALIM_BDD**
- pour objectif : Alimentation de la BDD OPENDATA_ASSEMBLEE_NATIONALE
- pour description le bloc suivant en remplaçant les {} par vos informations :

```
##########
DATE DE CREATION : {dd/MM/yyyy}
DATE DE MODIFICATION :
VERSION : 0.1
AUTEUR : {nom_prenom}
DESCRIPTION : CREATION DU JOB
##########
```

Étape 1 : Importer le groupe de contexte **DIR** dans le job **ALIM_BDD**

Étape 2 : Importer le groupe de contexte **P_VAR** dans le job **ALIM_BDD**

Étape 3 : Copier l'intégralité de **TEMPLATE_JOB** dans le job **ALIM_BDD**

Étape 4 : Spécifier une metadata de type **Connexions aux base de données** pour la BDD OPENDATA_ASSEMBLEE_NATIONALE avec pour paramétrage

```
NOM : BDD_ASSEMBLEE_NATIONALE
TYPE : POSTGRESQL
IDENTIFIANT : postgres
MDP : postgres
SERVEUR : OPENDATA_ASSEMBLEE_NATIONALE
PORT : 5432
ENCODAGE : UTF-8
DATABASE : OPENDATA_ASSEMBLEE_NATIONALE
```

- Créer un groupe de contextes associé à cette BDD

Étape 5 : Importer le groupe de contexte **BDD_ASSEMBLEE_NATIONALE** dans le job **ALIM_BDD**

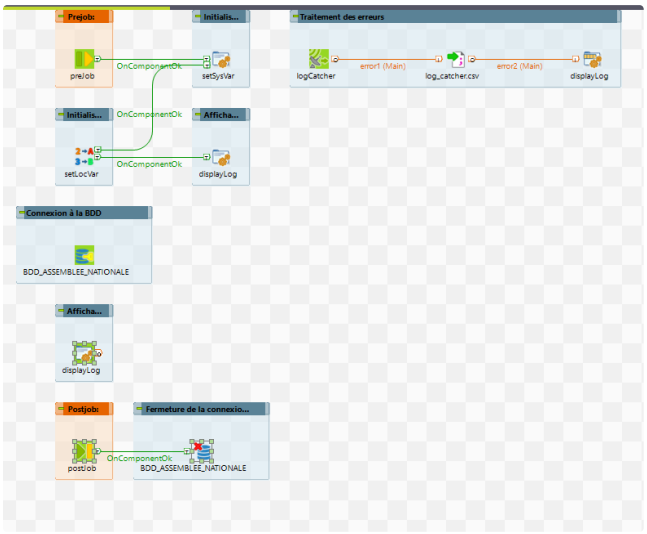
Étape 6 : Importer la métada **BDD_ASSEMBLEE_NATIONALE** en mode **tPostreSQLConnection** avec pour paramétrage

```
Nommage du composant : BDD_ASSEMBLEE_NATIONALE
Nommage du sous-job: Ouverture de la connexion à la BDD
##########
Paramètres avancés : Commit Automatique
```

Étape 7 : Dans le **tPostJob** ajouter un composant **tPostreSQLClose** qu'on lie en mode *OnComponentOk*

```
Nommage du composant : BDD_ASSEMBLEE_NATIONALE
Nommage du sous-job: Fermeture de la connexion à la BDD
##########
Connexion : BDD_ASSEMBLEE_NATIONALE
```

Votre job devrait ressembler à ça :



3.3.2.1 Sous-job d'alimentation des tables DEPUTE et COLLAB

- **tFileInputDelimited** : Ce composant lit un fichier délimité ligne par ligne, afin de le diviser en champs et d'envoyer ces champs au composant suivant, comme défini par le schéma.
- **tMap** : Ce composant transforme et route des données à partir d'une ou plusieurs source(s) de données vers une ou plusieurs destination(s).
- **tFileOutputDelimited** : Ce composant écrit en sortie les données d'entrée dans un fichier délimité en respectant le schéma défini.
- **tPostgreSQLOutput** : Ce composant écrit en sortie les données d'entrée dans une table de BDD de type PostgreSQL
- **tUniqRow**

Etape 0 : Spécifier une metadata de type **Fichier Délimité** avec pour paramétrage

```
NOM : FICHIER_DEPUTE
FICHIER : C:/workspace_talend/source/depute/depute.csv
FORMAT : UNIX
ENCODAGE : UTF-8
ENTOURAGE DU TEXTE : "\""
EN-TÊTE : 1
```

Nom du champ	Type	Nullable
id	STRING	YES

legislature	INTEGER	YES
civ	STRING	YES
nom	STRING	YES
prenom	STRING	YES
villeNaissance	STRING	YES
naissance	DATE ("dd-MM-yyyy")	YES
age	INTEGER	YES
groupe	STRING	YES
groupeAbrev	STRING	YES
departementNom	STRING	YES
departementCode	STRING	YES
datePriseFonction	STRING	YES
mail	STRING	YES
twitter	STRING	YES
facebook	STRING	YES
website	STRING	YES
nombreMandats	STRING	YES
experienceDepute	STRING	YES
scoreParticipation	STRING	YES
scoreParticipationSpecialite	STRING	YES
scoreLoyaute	STRING	YES
scoreMajorite	STRING	YES
dateMaj	STRING	YES

Nous avons dans un premier temps spécifié le type `STRING` pour l'ensemble des champs afin d'être sûr de bien pouvoir lire le fichier.

En effet, le type `STRING` est le plus complet dans le sens où touts les types peuvent être convertis en `STRING`
Voir Table 1

- Créer un groupe de contextes associé à ce fichier

Étape 1 : Spécifier une metadata de type **Fichier Délimité** avec pour paramétrage

```
NOM : FICHIER_DEPUTE
FICHIER : C:/workspace_talend/source/collaborateur/collaborateur.csv
FORMAT : UNIX
ENCODAGE : ISO-8859-1
ENTOURAGE DU TEXTE : "\""
EN-TËTE : 1
```

Nom du champ	Type	Nullable
id (Identifiant du député)	STRING	YES
nom_depute	STRING	YES
prenom_depute	STRING	YES
nom_collab	STRING	YES
prenom_collab	STRING	YES

- Créer un groupe de contextes associé à ce fichier

Étape 2 : Importer les métadas **FICHIER_DEPUTE & FICHIER_COLLAB** en mode **tFileInputDelimited**

Étape 3 : Importer les groupes de contextes **FICHIER_DEPUTE & FICHIER_COLLAB**

Étape 4 : Ajouter un composant **tMap** que l'on nomme **actionData** du côté droit du composant **FICHIER_DEPUTE** et au-dessus du composant **FICHIER_COLLAB**

Étape 5 : Cliquer droit sur le composant **FICHIER_DEPUTE** puis choisir un lien *Row>Main* que l'on nomme `depute_input` et le lier au composant **actionData**

Étape 6 : Dans le composant **actionData** créer une sortie `depute_transform` avec le mapping suivant :

Nom du champ cible	Alimentation	Nom du type de sortie
depute_code	StringHandling.UPCASE(StringHandling.TRIM(depute_input.id))	STRING
nom_depute	StringHandling.UPCASE(StringHandling.TRIM(depute_input.nom))	STRING
prenom_depute	StringHandling.UPCASE(StringHandling.TRIM(depute_input.prenom))	STRING
intitule_depute	StringHandling.UPCASE(StringHandling.TRIM(depute_input.civ + " " + depute_input.nom + " " + depute_input.prenom))	STRING
groupe_politique	StringHandling.UPCASE(StringHandling.TRIM(depute_input.groupeAbrev))	STRING
departement	StringHandling.UPCASE(StringHandling.TRIM(depute_input.departementNom))	STRING
circonscription	Integer.parseInt(depute_input.circo)	INTEGER

Étape 7 : Ajouter un composant **tUniqRow** en lien avec `depute_transfom` avec pour sortie `uniq_depute` et pour paramétrage

```
Nommage du composant : uniqDepute
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
CLE : depute_code
##########
```

Étape 8 : Ajouter un composant **tPostrgeSQLOutput e**n lien avec `uniq_depute` avec pour paramétrage

```
Nommage du composant : outputDepute
```

```
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
UTILISER LA CONNEXION BDD_ASSEMBLEE_NATIONALE
TABLE : DEPUTE
ACTION SUR LA TABLE : DEFAUT
ACTIO SUR LES DONNEES : INSERT OR UPDATE
SCHEMA : BUILT-IN EN MODE SYNCHRONISATION
CLE : depute_code
##########
```

Étape 9 : Cliquer droit sur le composant **FICHIER_COLLABORATEUR** puis choisir un lien *Row>Main* que l'on nomme `collab_input` et le lier au composant **actionData**

> Le lien `collab_input` devrait apparaitre en lien Lookup

Étape 10 : Dans le composant **actionData** créer une sortie `collab_transform` avec le mapping suivant :

Nom du champ cible	Alimentation	Nom du type de sortie
collab_code	Numeric.sequence(depute_input.id ,1,1)	INTEGER
depute_code	StringHandling.UPCASE(StringHandling.TRIM(collab_input.id))	STRING
nom_collab	StringHandling.UPCASE(StringHandling.TRIM(collab_input.nom_collab))	STRING
prenom_collab	StringHandling.UPCASE(StringHandling.TRIM(collab_input.prenom_collab))	STRING
intitule_collab	StringHandling.UPCASE(StringHandling.TRIM(collab_input.nom_collab + " " + depute_input.prenom_collab))	STRING

Étape 11 : Ajouter un composant **tPostrgeSQLOutput** en lien avec `collab_transform` avec pour paramétrage

```
Nommage du composant : outputCollab
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
UTILISER LA CONNEXION BDD_ASSEMBLEE_NATIONALE
TABLE : COLLAB
ACTION SUR LA TABLE : DEFAUT
ACTIO SUR LES DONNEES : INSERT OR UPDATE
SCHEMA : BUILT-IN EN MODE SYNCHRONISATION
CLE : depute_code ; collab_code
##########
```

Étape 12 : Dans le composant **actionData** créer une jointure de type INNER avec la relation `depute_input.id=collab_input.id` en gardant toute les correspondances

- Créer une sortie `rejet_jointure` avec le mapping suivant :

Nom du champ cible	Alimentation	Nom du type de sortie

depute_code_collab	collab_input.id	STRING
depute_code_depute	depute_input.id	STRING

Pour récupérer les rejets, il nécessaire d'activer le paramètre Catch lookup inner join reject

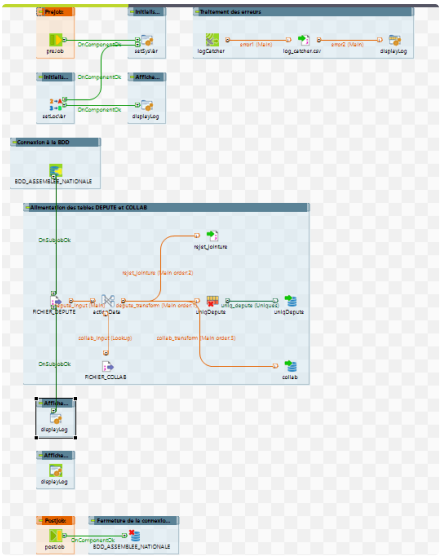
Étape 13 : Ajouter un composant **tFileOutputDelimited** en lien avec `rejet_jointure` avec pour paramétrage

```
Nommage du composant : rejet_jointure
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
NOM FICHIER : context.DIR_RACINE + "/cible/rejet_jointure.csv"
ECRIRE APRES
SEPARATEUR CHAMPS : ";"
SEPARATEUR LIGNES : "\n"
INCLURE L'EN-TÊTE
SCHEMA : depute_code_collab ; depute_code_depute
##########
```

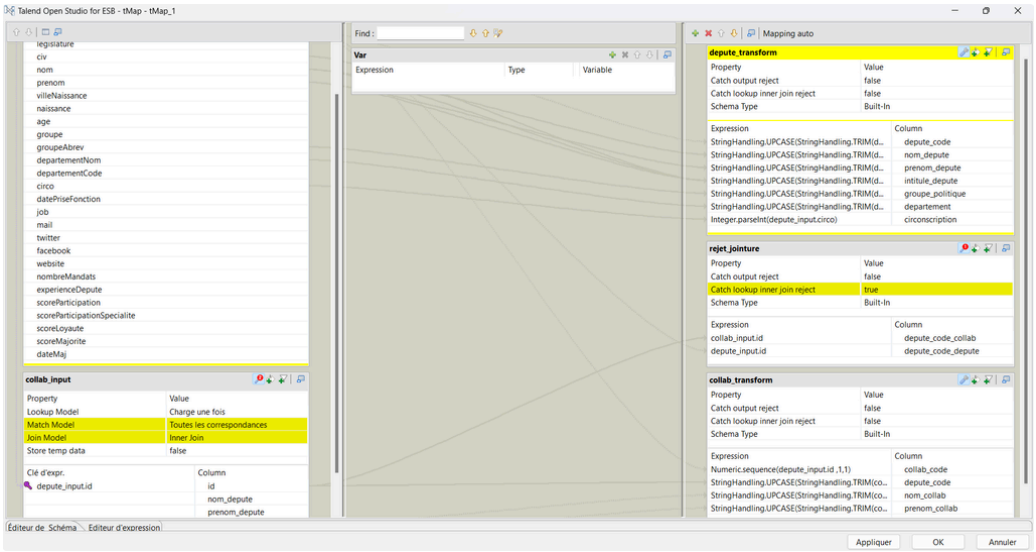
Étape 14 : Relier le sous-job à un composant **tJava** qu'on nomme **displayLog** par un lien de type *OnSubjobOk* avec le contenu suivant :

```
System.out.println("---------------");
System.out.println("ALIMENTATION DES TABLES COLLAB & DEPUTE : OK ");
System.out.println("---------------");
System.out.println("TABLE DEPUTE :");
System.out.println("NB LIGNE INSERTED : " + (Integer)globalMap.get("tDBOutput_1_NB_LINE_INSERTED"));
System.out.println("NB LIGNE UPDATED : " + (Integer)globalMap.get("tDBOutput_1_NB_LINE_UPDATED"));
System.out.println("---------------");
System.out.println("TABLE COLLAB :");
System.out.println("NB LIGNE INSERTED : " + (Integer)globalMap.get("tDBOutput_2_NB_LINE_INSERTED"));
System.out.println("NB LIGNE UPDATED : " + (Integer)globalMap.get("tDBOutput_2_NB_LINE_UPDATED"));
```

Votre job devrait ressembler à ça :



Et plus particulièrement pour le **tMap(actionData)**



3.3.2.2 Sous-job d'alimentation des tables LOI et VOTE

- **tJavaRow**: Ce composant exécute du code Java sur chaque ligne
- **tFileInputJson**: Ce composant lit un fichier de type JSON.
- **tUnite :** Ce composant merge deux dataset

Étape 0 : Modifier les propriétés du flux **ALIM_BDD** :

- ajouter dans la description le bloc suivant en remplaçant les {} par vos informations :

```
##########
DATE DE CREATION : {dd/MM/yyyy}
DATE DE MODIFICATION : {dd/MM/yyyy}
VERSION : 0.2
AUTEUR : {nom_prenom}
DESCRIPTION : Ajout du sous-job d'alimentation des votes et de la loi
##########
```

- incrémenter la version en 0.2

Étape 1 : Ajouter un composant **tFileList** en lien *OnSubjobOK* avec le composant **displayLog** de fin du sous-job précédent

```
Nommage du composant : listeLoi
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
NOM DU DOSSIER : context.DIR_RACINE + "/source/vote/json/"
##########
```

Étape 2 : Relier le **tFileList** à un composant **tJava** qu'on nomme **displayLog** par un lien de type *Iterate* avec le contenu suivant :

```
System.out.println("---------------");
System.out.println("TRAITEMENT DE LA LOI : ");
System.out.println((String)globalMap.get("tFileList_1_CURRENT_FILE"));
```

Étape 3 : Ajouter un composant **tFileInputJSON** en lien *OnComponentOk* avec le précédent composant **displayLog**

```
Nommage du composant : FICHIER_POUR
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
LU PAR : JSONPATH
LOOP JSON QUERY : "$.scrutin.ventilationVotes.organe.groupes.groupe[*].vote.decompteNominatif.pours.votant[*]"
```

```
SCHEMA :
- depute_code de type STRING avec pour requête JSON "acteurRef"
- intitule_loi de type STRING avec pour requête JSON "$.scrutin.titre"
NOM DU FICHIER : ((String)globalMap.get("tFileList_1_CURRENT_FILEPATH"))
##########
```

Étape 4 : Relier le **FICHIER_POUR** à un composant **tJavaRow** qu'on nomme **transcation_set** par un lien de type *Main* que l'on appelle `pour` avec le contenu suivant :

```
output_pour.depute_code = output_pour.depute_code ;
output_pour.loi_code = (String)globalMap.get("tFileList_1_CURRENT_FILE") ;
outpt_pour.vote_code = 1;
outup_pour.intitule_loi = StringHandling.LEFT(pour.intitule_loi,1000);
```

Étape 5 : Ajouter un composant **tFileInputJSON**

```
Nommage du composant : FICHIER_CONTRE
Nommage du sous-job: Alimentation des tables DEPUTE et COLLAB
##########
LU PAR : JSONPATH
LOOP JSON QUERY : "$.scrutin.ventilationVotes.organe.groupes.groupe[*].vote.decompteNominatif.contres.votant[*]"
SCHEMA :
- depute_code de type STRING avec pour requête JSON "acteurRef"
- intitule_loi de type STRING avec pour requête JSON "$.scrutin.titre"
NOM DU FICHIER : ((String)globalMap.get("tFileList_1_CURRENT_FILEPATH"))
##########
```

Étape 6 : Relier le **FICHIER_CONTRE** à un composant **tJavaRow** qu'on nomme **transcation_set** par un lien de type *Main* que l'on appelle `contre` avec le contenu suivant :

```
output_contre.depute_code = contre.depute_code;
output_contre.loi_code = (String)globalMap.get("tFileList_1_CURRENT_FILE") ;
output_contre.vote_code = 0;
output_contre.intitule_loi = StringHandling.LEFT(contre.intitule_loi,1000);
```

Étape 7 : Relier chaque **transcation_set** à un composant **tUnite** avec pour lien *Main,* respectivement nommé :

- `output_contre` pour la branche CONTRE
- `output_pour` pour la branche POUR

Nom du champ	Nom du type de sortie
depute_code	STRING
loi_code	STRING
vote_code	INTEGER
intitule_loi	STRING

Nous appelons ce composant **uniteVote**

Étape 8 : Ajouter un composant **tMap** que l'on nomme **actionData** en lien *Main* avec **uniteVote ;** lien que l'on nomme `data`

Étape 9 : Dans le composant **actionData** créer une sortie `vote_transform` avec le mapping suivant :

Nom du champ cible	Alimentation	Nom du type de sortie
depute_code	data.depute_code	STRING

loi_code	data.loi_code	STRING
vote_code	data.vote_code	INTEGER

On ajoute également pour cette sortie le filtre suivant :

```
data.depute_code != null && data.depute_code != "null"
```

Étape 10 : Ajouter un composant **tPostrgeSQLOutput** en lien avec `vote_transform` avec pour paramétrage

```
Nommage du composant : outputVote
Nommage du sous-job: Alimentation des tables VOTE et DEPUTE
##########
UTILISER LA CONNEXION BDD_ASSEMBLEE_NATIONALE
TABLE : VOTE
ACTION SUR LA TABLE : DEFAUT
ACTIO SUR LES DONNEES : INSERT OR UPDATE
SCHEMA : BUILT-IN EN MODE SYNCHRONISATION
CLE : depute_code ; loi_code
##########
```

Étape 11 : Dans le composant **actionData** créer une sortie `loi_transform` avec le mapping suivant :

Nom du champ cible	Alimentation	Nom du type de sortie
loi_code	data.loi_code	STRING
intitule_loi	data.intitule_loi	STRING

Étape 12 : Ajouter un composant **tUniqRow** en lien avec `loi_transform` ayant pour sortie `uniq_loi` et pour paramétrage

```
Nommage du composant : uniqLoi
Nommage du sous-job: Alimentation des tables VOTE et DEPUTE
##########
CLE : loi_code
##########
```

Étape 13 : Ajouter un composant **tPostrgeSQLOutput** en lien avec `uniq_loi` avec pour paramétrage

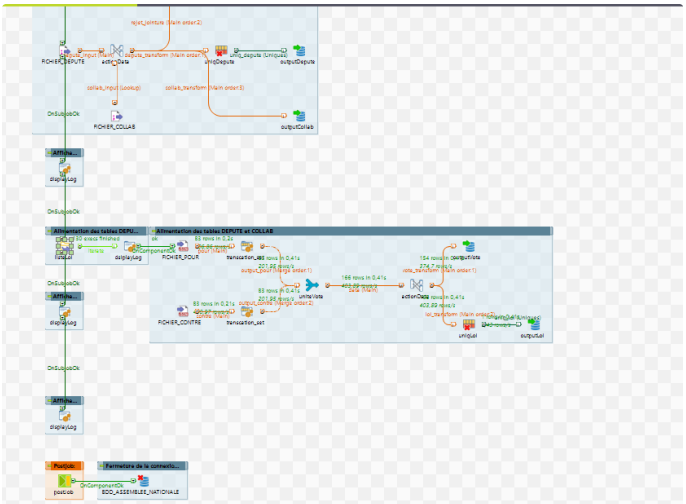
```
Nommage du composant : outputLoi
Nommage du sous-job: Alimentation des tables VOTE et DEPUTE
##########
UTILISER LA CONNEXION BDD_ASSEMBLEE_NATIONALE
TABLE : VOTE
ACTION SUR LA TABLE : DEFAUT
ACTIO SUR LES DONNEES : INSERT OR UPDATE
SCHEMA : BUILT-IN EN MODE SYNCHRONISATION
CLE : loi_code
##########
```

Étape 14 : Relier le sous-job à un composant **tJava** qu'on nomme **displayLog** par un lien de type *OnSubjobOk* avec le contenu suivant :

```
System.out.println("---------------");
System.out.println("ALIMENTATION DES TABLES VOTE & LOI: OK ");
System.out.println("---------------");
```

Étape 15 : Relier le précédent **tJava(displayLog)** au dernier **tJava(displayLog)** issue du **TEMPLATE_JOB**

Votre job devrait ressembler à ça :



3.2.3 Exécution du job

Étape 0 : Pour pouvoir exécuter un job, il convient de se mettre dans l'onglet "EXÉCUTER" dans le QUADRANT SUD EST .

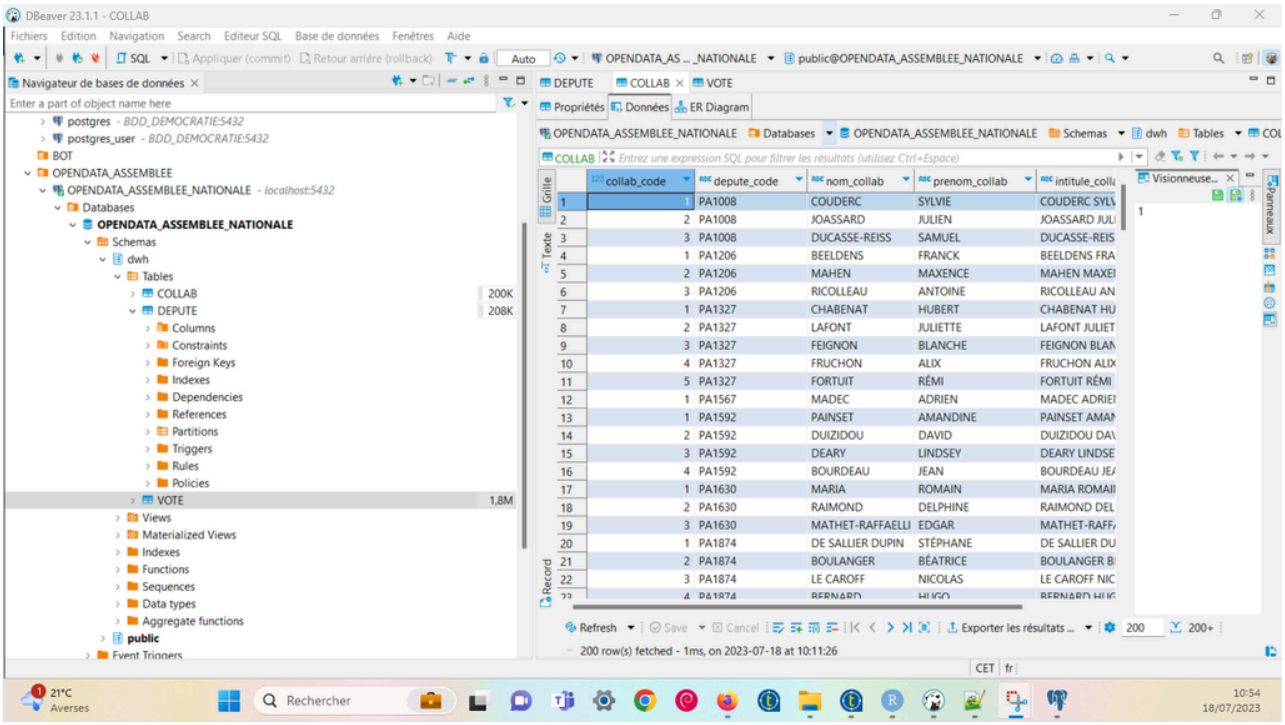
Étape 1 : Cliquer EXÉCUTER en utilisant l'environnement **DEV** et observer qu'aucune erreur de compilation n'est générée.

> Le traitement peut être assez long, vous pouvez tranquillement patienter en allant vous cherchez un ☕

Dans cette partie, nous allons nous assurer que le job a bien fait son travail d'alimentation.
Pour cela nous allons utiliser **DBeaver**.

> Nous aurions pus nous contenter de **PgAdmin** cependant cette outil ne permet d'interagir qu'avec des base de données de type PostgreSQL, là où **DBeaver** permet d'interagir avec un grand nombre d'autre type.

- Ouvrir **Dbeaver**
- Créer un dossier **OPENDATA_ASSEMBLEE**
- Créer une connexion PostgreSQL à la BDD **OPENDATA_ASSEMBLEE_NATIONALE**
- Double-cliquer sur les tables afin de visualiser les données



3.4 Récupération des informations

3.4.1 Création du service GET_INFO_BDD

Nous allons maintenant créer le service REST.

- **tRestRequest** : Pour définir la requête REST que le client doit appeler
- **tPostgreSQLInput** : Table de la base de données
- **tFlowToIterate** : Pour effectuer une itération sur les données d'entrée et générer des variables globales.
- **tXMLMap** : Permet de router et transformer les flux entrants de la base de données vers le résultat de la requête.
- **tRestResponse** : Pour définir la réponse à envoyer à l'utilisateur suite à sa requête.

Étape 0 : Créer un job avec :

- pour titre : **GET_INFO_BDD**
- pour objectif : Récupération des informations via un service REST
- pour description le bloc suivant en remplaçant les {} par vos informations :

```
##########
DATE DE CREATION : {dd/MM/yyyy}
DATE DE MODIFICATION :
VERSION : 0.1
AUTEUR : {nom_prenom}
DESCRIPTION : CREATION DU SERVICE
##########
```

Étape 1 : Importer le groupe de contexte **BDD_ASSEMBLEE_NATIONALE** dans le job **ALIM_BDD**

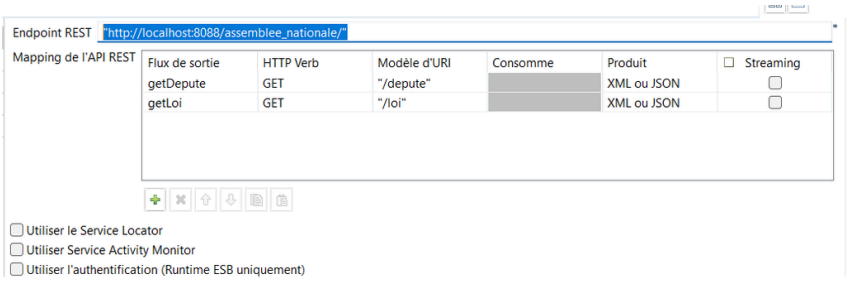
Étape 2 : Importer la métada **BDD_ASSEMBLEE_NATIONALE** en mode **tPostreSQLConnection** avec pour paramétrage

```
Nommage du composant : BDD_ASSEMBLEE_NATIONALE
Nommage du sous-job: Ouverture de la connexion à la BDD
```

```
##########
Paramètres avancés : Commit Automatique
```

Étape 3 : Ajouter un composant **tRestRequest** avec pour paramétrage

```
Nommage du composant : getData
Nommage du sous-job: Obtention des informations de la BDD OPENDATA_ASSEMBLEE_NATIONALE
```



- En cliquant sur *getDepute*, un bouton avec trois petits points apparaît. Cliquez dessus.
  - Ajouter les deux colonnes *depute_code* représentant le paramètre de la requête. Prenez soin à ce que:
    - Son type soit *string*
    - La valeur par défaut soit de PA722142 .
    - Leur commentaire ait la valeur: *query*
- En cliquant sur *getLoi* , un bouton avec trois petits points apparaît. Cliquez dessus.
  - Ajouter la colonne *loi_code* représentant le paramètre de la requête. Prenez soin à ce que:
    - Son type soit *string*
    - La valeur par défaut soit de VTANR5L16V3213.json
    - Leur commentaire ait la valeur: *query*

> La valeur par défaut est utilisée dans le cas où le consommateur n'introduit pas de paramètres.

> Le commentaire query indique que ce champ est un paramètre de requête, pas définie dans le Path.

Nous désirons configurer le service de manière à ce que, quand un consommateur appelle :

- l'URI `http://localhost:8088/assemblee_nationale/depute?depute_code=param` avec pour `param` , un identifiant de `depute` , le service retourne une réponse contenant les informations du député dont une liste des collaborateur.
- l'URI `http://localhost:8088/assemblee_nationale/loi?loi_code=param` avec pour `param` , un identifiant de `loi` , le service retourne une réponse contenant les informations de la loi et les votes associés.

3.4.1.1 Sous-service getDepute

Étape 0 : Relier la sortie `getDepute` à un composant **tFlowToIterate** qu'on nomme **deputeResponse**

Étape 1 : Ajouter un composant **tPostrgeSQLInput** en lien *Iterate* , avec pour sortie `data_depute` et pour paramétrage

```
Nommage du composant : getDataDepute
Nommage du sous-job: Obtention des informations de la BDD OPENDATA_ASSEMBLEE_NATIONALE
##########
UTILISER LA CONNEXION BDD_ASSEMBLEE_NATIONALE
##########
```

```
"
SELECT

 D.depute_code,
```

```
 D.nom_depute,
 D.prenom_depute,
 D.intitule_depute ,
 D.groupe_politique,
 C.collab_code,
 C.nom_collab,
 C.prenom_collab,
 C.intitule_collab

FROM dwh.\"DEPUTE\" D

INNER JOIN dwh.\"COLLAB\" C
ON D.depute_code = C.depute_code


WHERE D.depute_code IN ( '"+ globalMap.get("getDepute.depute_code") + "' )
"
```

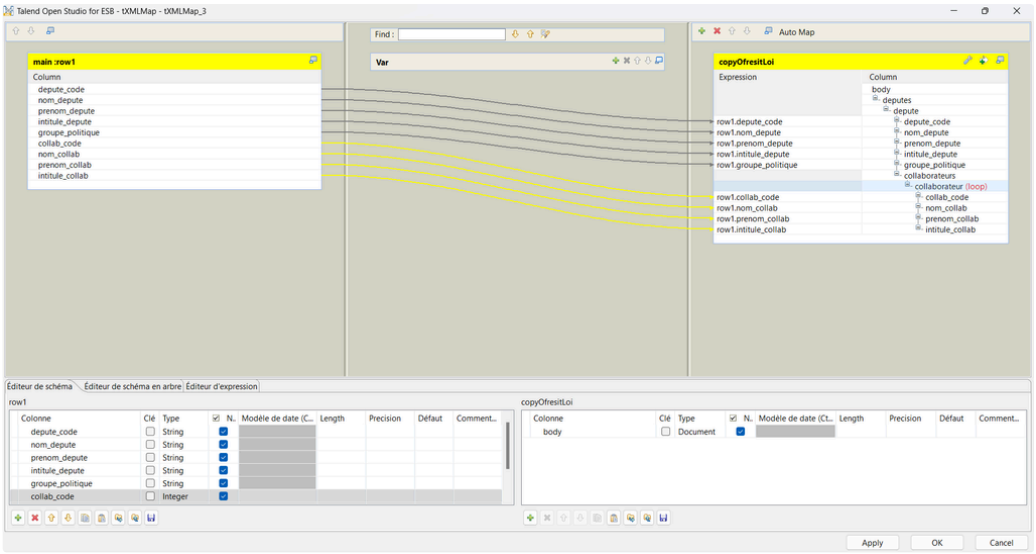
Nom du champ	Type	Nullable
depute_code	STRING	YES
nom_depute	STRING	YES
prenom_depute	STRING	YES
intitule_depute	STRING	YES
collab_code	INTEGER	YES
nom_collab	STRING	YES
prenom_collab	STRING	YES
intitule_collab	STRING	YES

Étape 3 : Ajouter un composant **tXMLMap(restitDepute)** configurer comme suit

Cliquer deux fois sur le **tXMLMap** pour la configurer.

- Ajouter une sortie `restitDepute`
- Dans la colonne de droite, ajouter (si ce n'est déjà fait) une colonne intitulée *body* dont le type est *Document*.
- Cette colonne contient un élément *root*. Renommer cet élément pour *deputes* .
- Ajouter un sous-élément à *deputes* appelé *depute*
  - Glisser-déplacer le *prenom_depute*,*prenom_depute*,*groupe_politique*,*intitule_depute* des colonnes en entrée vers le *depute*. Créez-le comme sous-éléments du noeud cible.
- Ajouter un sous-élément à *depute* appelé *collaborateurs*.
- Ajouter un sous-élément à *collaborateurs* appelé *collaborateur*.
  - Définir cet élément comme *loop Element*.
  - Glisser-déplacer le *collab_code*,*prenom_collab*,*prenom_collab*,*intitule_collab* des colonnes en entrée vers le *collaborateur*. Créez-le comme sous-éléments du nœud cible.
- Dans la colonne de droite, cliquer sur la petite clef à molette
  - Mettre la valeur de "All in one" à *true*. Cela permettra à toutes les données XML d'être écrites dans un seul flux.

La configuration finale sera donc comme suit:

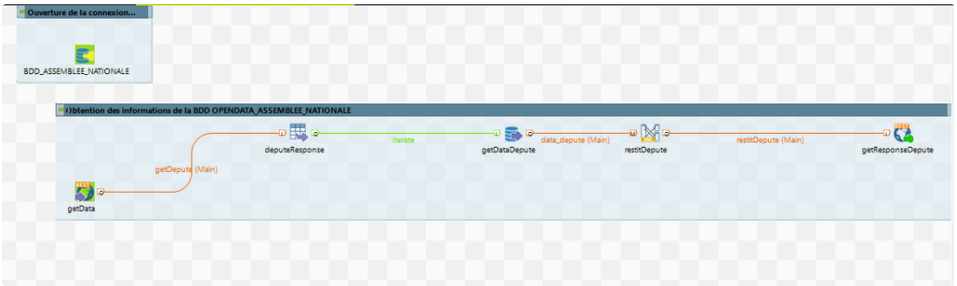


La configuration précédente va générer une réponse de la forme suivante:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<deputes>
    <depute>
        <depute_code>PA795596</nom_depute>
        <nom_depute>VANNIER</nom_depute>
        <prenom_depute>PAUL</prenom_depute>
        <intitule_depute>M. VANNIER PAUL</intitule_depute>
        <groupe_politique>LFI-NUPES</groupe_politique>
        <collaborateurs>
            <collaborateur>
                <collab_code>1</collab_code>
                <nom_collab>MARZOUGUI</nom_collab>
                <prenom_collab>ANIS</prenom_collab>
                <intitule_collab>MARZOUGUI ANIS</intitule_collab>
            </collaborateur>
            <collaborateur>
                <collab_code>2</collab_code>
                <nom_collab>FOUCAULT</nom_collab>
                <prenom_collab>CLÉMENCE</prenom_collab>
                <intitule_collab>FOUCAULT CLÉMENCE</intitule_collab>
            </collaborateur>
            <collaborateur>
                <collab_code>3</collab_code>
                <nom_collab>AWAD ABDOU</nom_collab>
                <prenom_collab>MOHAMED</prenom_collab>
                <intitule_collab>AWAD ABDOU MOHAMED</intitule_collab>
            </collaborateur>
        </collaborateurs>
    </depute>
</deputes>
```

Étape 3 : Ajouter un composant **tRestResponse** que l'on nomme **getResponseDepute** en lien `resititDepute` que l'on laisse par défaut

Votre job devrait ressembler à ça :



3.4.1.2 Sous-service getLoi

Étape 0 : Relier la sortie `getLoi` à un composant **tFlowToIterate** qu'on nomme **loiResponse**

Étape 1 : Ajouter un composant **tPostrgeSQLInput** en lien *Iterate* , avec pour sortie `data_loi` et pour paramétrage

```
Nommage du composant : getDataLoi
Nommage du sous-job: Obtention des informations de la BDD OPENDATA_ASSEMBLEE_NATIONALE
##########
UTILISER LA CONNEXION BDD_ASSEMBLEE_NATIONALE
##########
```

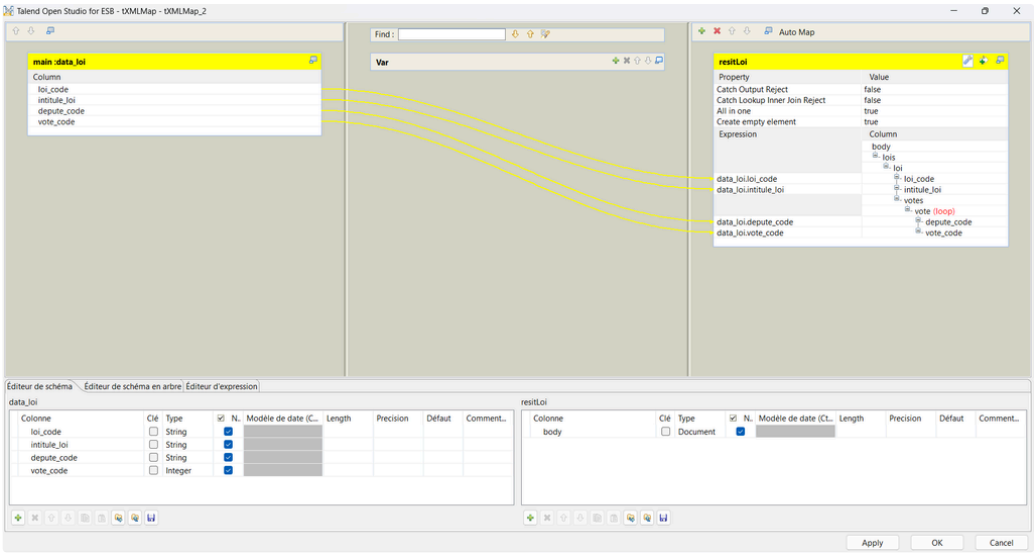
```
"
SELECT

 L.loi_code,
 L.intitule_loi,
 V.depute_code,
 V.vote_code

FROM dwh.\"LOI\" L

LEFT JOIN dwh.\"VOTE\" V
ON L.loi_code= V.loi_code
"
```

Étape 3 : Ajouter un composant **tXMLMap(restitLoi**) configurer comme suit

Cliquer deux fois sur le **tXMLMap** pour la configurer.

- Ajouter une sortie `restitLoi`
- Dans la colonne de droite, ajouter (si ce n'est déjà fait) une colonne intitulée *body* dont le type est *Document*.
- Cette colonne contient un élément *root*. Renommer cet élément pour *lois*.
- Ajouter un sous-élément à *lois* appelé *loi*
  - Glisser-déplacer le *loi_code*,*intitule_loi* des colonnes vers la *loi*. Créez-le comme sous-éléments du noeud cible.
- Ajouter un sous-élément à *loi* appelé *votes*.
- Ajouter un sous-élément à *votes* appelé *vote* .
  - Définir cet élément comme *loop Element*.
  - Glisser-déplacer le *depute_code*,*vote_code*,des colonnes en entrée vers le *vote* . Créez-le comme sous-éléments du nœud cible.
- Dans la colonne de droite, cliquer sur la petite clef à molette
  - Mettre la valeur de "All in one" à *true*. Cela permettra à toutes les données XML d'être écrites dans un seul flux.

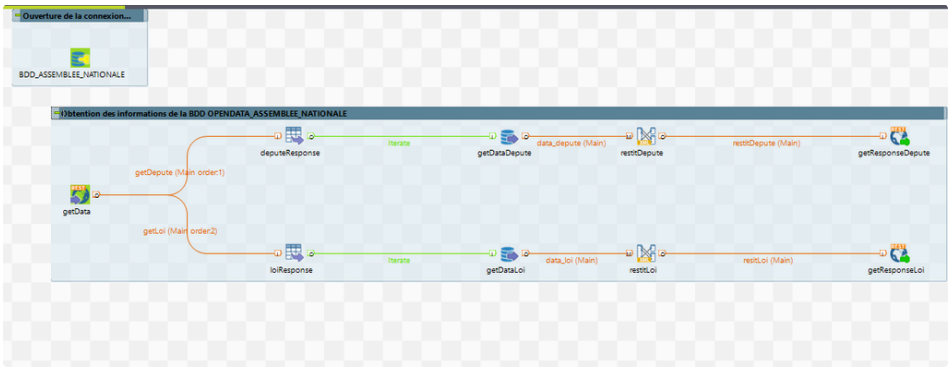
La configuration finale sera donc comme suit:



La configuration précédente va générer une réponse de la forme :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<lois>
    <loi>
        <loi_code>VTANR5L16V1.json</loi_code>
        <intitule_loi>la motion de censure déposée en application de l'article 49, alinéa 2, de la Consti
        <votes>
            <vote>
                <depute_code>PA720892</depute_code>
                <vote_code>1</vote_code>
            </vote>
            <vote>
                <depute_code>PA721062</depute_code>
                <vote_code>1</vote_code>
            </vote>
        </votes>
    <loi>
<lois>
```

Votre job devrait ressembler à ça :



3.4.3 Test du service

Étape 0 : Se mettre dans l'onglet "EXÉCUTER" dans le QUADRANT SUD EST .

Étape 1 : Cliquer EXÉCUTER en utilisant l'environnement **DEV**

Tant que vous n'arrêté pas le service, celui-ci continue de tourner

Étape 2 :

Pour tester le service, il suffit d'ouvrir un navigateur, et de taper la requête de votre choix.

- Par exemple, la requête suivante : `http://localhost:8088/assemblee_nationale/depute?depute_code=PA795596` donnera:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<deputes>
  <depute depute_code="PA795596">
    <nom_depute>VANNIER</nom_depute>
    <prenom_depute>PAUL</prenom_depute>
    <intitule_depute>M. VANNIER PAUL</intitule_depute>
    <groupe_politique>LFI-NUPES</groupe_politique>
    <collaborateurs>
      <collaborateur collab_code="1">
        <nom_collab>MARZOUGUI</nom_collab>
        <prenom_collab>ANIS</prenom_collab>
        <intitule_collab>MARZOUGUI ANIS</intitule_collab>
      </collaborateur>
      <collaborateur collab_code="2">
        <nom_collab>FOUCAULT</nom_collab>
        <prenom_collab>CLÉMENCE</prenom_collab>
        <intitule_collab>FOUCAULT CLÉMENCE</intitule_collab>
      </collaborateur>
      <collaborateur collab_code="3">
        <nom_collab>AWAD ABDOU</nom_collab>
        <prenom_collab>MOHAMED</prenom_collab>
        <intitule_collab>AWAD ABDOU MOHAMED</intitule_collab>
      </collaborateur>
    </collaborateurs>
  </depute>
</deputes>
```

- Par exemple, la requête suivante : `http://localhost:8088/assemblee_nationale/loi` donnera:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<lois>
    <loi>
        <loi_code>VTANR5L16V1.json</loi_code>
        <intitule_loi>la motion de censure déposée en application de l'article 49, alinéa 2, de la Constitution
        <votes>
            <vote>
                <depute_code>PA720892</depute_code>
                <vote_code>1</vote_code>
            </vote>
            <vote>
                <depute_code>PA721062</depute_code>
                <vote_code>1</vote_code>
            </vote>
        </votes>
    <loi>
<lois>
```

Étape 3 :

Il est possible de tester votre service REST avec POSTMAN qui vous permettra de créer des collections d'appels API :

- Lancer **POSTMAN**

- Créer un workspace avec :
  - pour titre : **OPENDATA_ASSEMBLEE_NATIONALE**
  - pour description : Collection d'API lié à l'Open Data de l'Assemblée Nationale
- Créer une collection **COLLECTION_GET**
- Créer un appel **GET_DEPUTE**
  - Entrer l'URI que vous désirez tester: http://localhost:8088/assemblee_nationale/depute?depute_code=PA795596
  - Cliquer sur SEND.
- Créer un appel **GET_LOI**
  - Entrer l'URI que vous désirez tester: http://localhost:8088/assemblee_nationale/loi
  - Cliquer sur SEND.

Étape 4 :

Pour créer un consommateur pour le web service REST avec Talend, il suffit de créer le Job suivant:

3.5 Intégration des données

Nous allons maintenant créer une route permettant d'intégrer des nouveaux votes.

3.5.1 Création d'une route post_vote

Nous allons maintenant créer une route basé sur la framework Camel

- **cHTTP** : Ce composant fournit des endpoints basés HTTP pour consommer et produire des ressources HTTP
- **cSetHeader** : Ce composant définit des en-têtes ou personnalise les en-têtes par défaut, s'il y en a, dans chaque message qui lui est envoyé pour traitement subséquent du message.
- **cBean** : Ces propriétés sont utilisées pour configurer le cBean s'exécutant dans le framework de Jobs Standard.
- **cMessagingEndpoint** : Ce composant permet à deux applications de communiquer en envoyant ou en recevant des messages.
- **cDirect** : Pour définir la réponse à envoyer à l'utilisateur suite à sa requête.

Étape 0 : Créer une route avec :

- pour titre : **post_vote**
- pour objectif : Récupération de nouveau vote
- pour description le bloc suivant en remplaçant les {} par vos informations :

```
##########
DATE DE CREATION : {dd/MM/yyyy}
DATE DE MODIFICATION :
VERSION : 0.1
AUTEUR : {nom_prenom}
DESCRIPTION : CREATION DE LA ROUTE
##########
```

Étape 2 :

- Importer la route template_route.zip
- Créer les variable de contexte suivante dans le job **post_vote**
  - work_queue_cortex avec pour valeur http://0.0.0.0:8041/assemblee_nationale
  - work_queue_dead avec pour valeur Q.DEAD.VOTE

Étape 3 : Copier l'intégralité de **template_route** dans la route **POST_VOTE** et notamment le **Beans**

- JSONToHeadersBean 0.1

Étape 4 : Ajouter un composant **cHTTP** avec pour paramétrage

```
Nommage du composant : post
##########
MODE : SERVEUR
URI : context.work_http_consume
##########
```

Étape 5 : Ajouter un composant **cSetHeader** avec pour paramétrage

```
Nommage du composant : Cortex
##########
"businessName" avec pour language Constant et pour valeur jobName
"businessStartDate" avec pour language Simple et pour valeur "${date:now:yyyy-MM-dd'T'HH:mm:ss.SSSZ}"
##########
```

Étape 6 : Ajouter un composant **cBean** avec pour paramétrage

```
Nommage du composant : getData
##########
MODE : REFERENCE
ID "JSONToHeadersBean":
METHODE : "jsonToHeaders"
##########
```

On va notamment appliquer le code suivant afin de récupérer les informations du body pour les mettre dansle header afin de les injecter dans une requête SQL

```java
package beans;

import org.apache.camel.Headers;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
/*
 * user specification: the function's comment should contain keys as follows: 1. write about the function
 * it must be before the "{talendTypes}" key.
 *
 * 2. {talendTypes} 's value must be talend Type, it is required . its value should be one of: String, cha
 * long | Long, int | Integer, boolean | Boolean, byte | Byte, Date, double | Double, float | Float, Objec
 * Short
 *
 * 3. {Category} define a category for the Function. it is required. its value is user-defined .
 *
```

```
 * 4. {param} 's format is: {param} <type>[(<default value or closed list values>)] <name>[ : <comment>]
 *
 * <type> 's value should be one of: string, int, list, double, object, boolean, long, char, date. <name>
 * Function's parameter name. the {param} is optional. so if you the Function without the parameters. the
 * added. you can have many parameters for the Function.
 *
 * 5. {example} gives a example for the Function. it is optional.
 */
public class JSONToHeadersBean  {

    public void jsonToHeaders(String body, @Headers Map<String, String> headers) throws ParseException {
        JSONParser parser = new JSONParser();
        JSONObject object = (JSONObject) parser.parse(body);
        object.keySet().forEach(key -> headers.put(key.toString(), object.get(key).toString()));
    }

    //for test
    public static void main(String[] args) throws ParseException {
        String body = "{\"msgId\": \"8600C5A3-C666-4E63-BFDB-52BCF557F938\", \"jiraId\": \"ERR002\"}";
        JSONParser parser = new JSONParser();
        JSONObject object = (JSONObject) parser.parse(body);
        final Map<String, String> headers = new HashMap<String, String>();
        object.keySet().forEach(key -> headers.put(key.toString(), object.get(key).toString()));
        System.out.println();
    }
}

```

Étape 7 : Ajouter un composant **cMessagingEndpoint** avec pour paramétrage

```
Nommage du composant : insertVote
##########
URI : "sql:insert into dwh.\"VOTE\" (depute_code,loi_code,vote_code) values (:#depute_code, :#loi_code, :#vote_co
DEPENDANCE : SQL
##########
```

Étape 9 : Ajouter un composant **cSetHeader** avec pour paramétrage

```
Nommage du composant : Cortex OK
##########
"businessStatus" avec pour language Constant et pour valeur "OK"
##########
```

Étape 9 : Ajouter un composant **cDirect** avec pour paramétrage

```
Nommage du composant : @toCortex
##########
Rediriger vers le cDirect @toCortex
##########
```

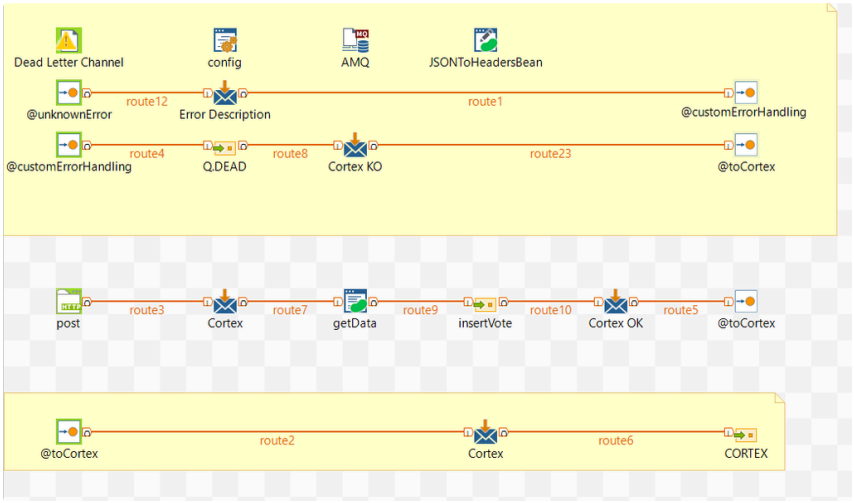
Étape 8 : Dans la partie Spring, rajoutter le code suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Used to inject external resources, beans or define more CamelContext and RouteBuilder here-->
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.
           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-co
           http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
           http://www.springframework.org/schema/jdbc/ http://www.springframework.org/schema/jdbc/spring-jdbc.xs
```

```
    <bean id="jmxEventNotifier" class="org.apache.camel.management.JmxNotificationEventNotifier">
        <property name="source" value="sdc"/>
        <property name="ignoreCamelContextEvents" value="true"/>
        <property name="ignoreRouteEvents" value="true"/>
        <property name="ignoreServiceEvents" value="true"/>
        <property name="ignoreExchangeEvents" value="true"/>
    </bean>
   <bean id="opendata_assemblee_nationale" class="org.postgresql.ds.PGPoolingDataSource" destroy-method="close">
      <property name="serverName" value="localhost"/>
      <property name="databaseName" value="OPENDATA_ASSEMBLEE_NATIONALE"/>
      <property name="user" value="postgres"/>
      <property name="password" value="postgres"/>
  </bean>
    <bean id="jmsFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
        <property name="brokerURL">
            <value>tcp://localhost:61616</value>
        </property>
    </bean>
</beans>
```

Spring est un framework de développement d'applications pour la version Enterprise de Java. Apache Camel est conçu pour fonctionner en harmonie avec le framework Spring. Le Route Designer du Studio Talend vous permet d'ajouter un contexte Spring à une Route pour un objectif de configuration. Vous pouvez définir des beans et des ressources en Spring XML DSL et les utiliser dans des Routes. Cela permet aux développeurs de combiner des codes Java et Spring dans la configuration des Routes. Cela s'avère utile lorsqu'il n'y a pas de composant explicite disponible dans la Palette.

Votre route devrait ressembler à ça :



3.5.2 Test de la route

Apache ActiveMQ est un courtier (broker) de messages open source écrit en Java avec un client complet de Java Message Service.
Il fournit des fonctionnalités pour les entreprises , comme de simplifier et encourager la communication des applications par messages

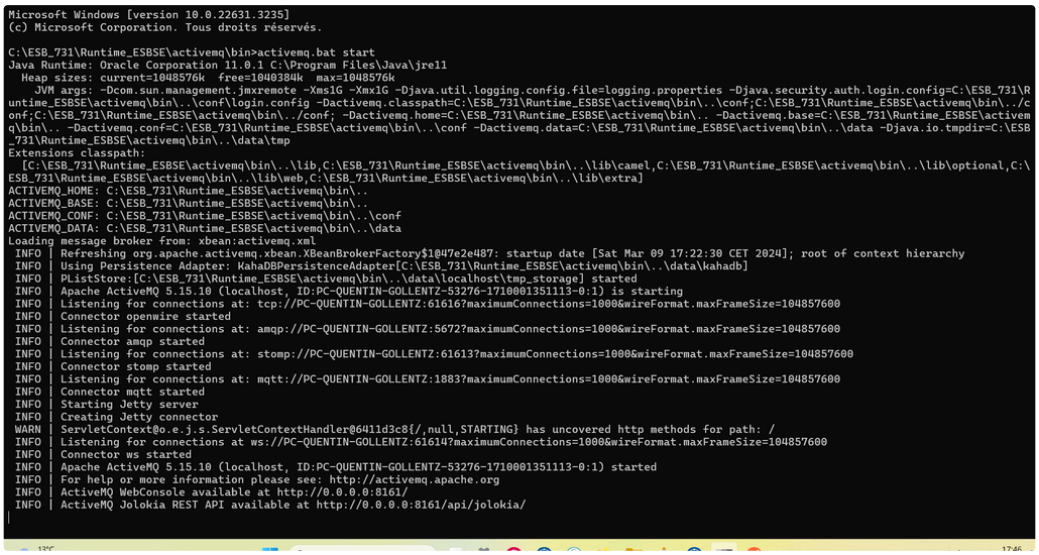
http://localhost:8161 avec :

- user : admin
- pwd : admin

Étape 0 : Se mettre dans le dossier *C:\ESB_731\Runtime_ESBSE\activemq\bin*

Étape 1 : Lancer la commande

```
activemq.bat start
```



Étape 2 : Se mettre dans l'onglet "EXÉCUTER" dans le QUADRANT SUD EST .

Étape 3 : Cliquer EXÉCUTER en utilisant l'environnement **DEV**

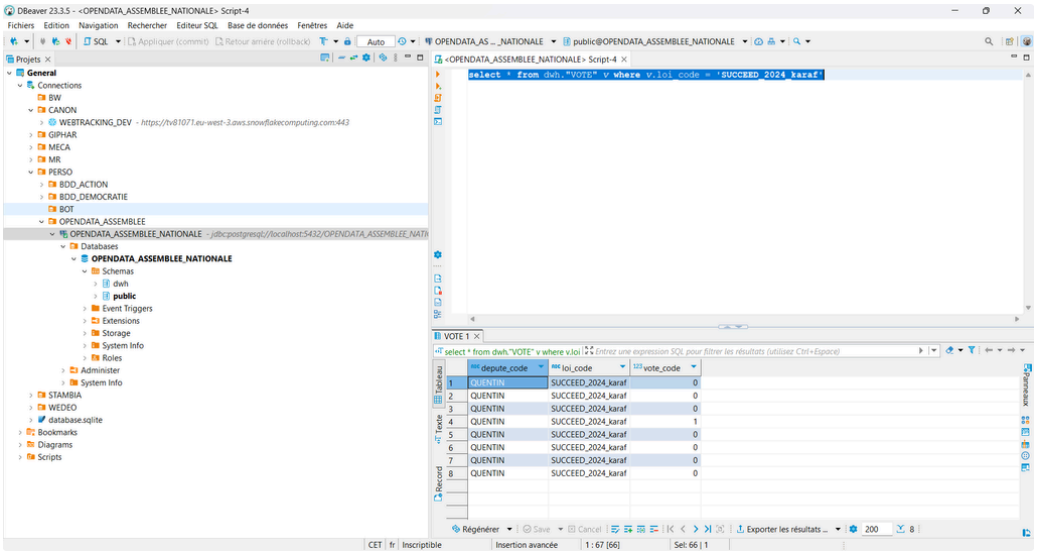
Étape 4 :

- Lancer **POSTMAN**
- Dans le workspace **OPENDATA_ASSEMBLEE_NATIONALE**
  - Créer une collection **COLLECTION_POST**
- Créer un appel **POST_VOTE** en méthode POST
  - Entrer l'URI que vous désirez tester: http://0.0.0.0:8041/assemblee_nationale
  - Pour body :
    - ```
      {
          "loi_code": "SUCCEED_2024_karaf",
          "vote_code": "0",
          "depute_code": "QUENTIN"
      }
      ```
- Cliquer sur SEND.

Étape 4 :

- Ouvrir **Dbeaver**
- Créer un script lié à la connexion PostgreSQL à la BDD **OPENDATA_ASSEMBLEE_NATIONALE**
- Lancer le script suivant :

```
select * from dwh."VOTE" v where v.loi_code = 'SUCCEED_2024_karaf'
```
```



4. Construction et déploiement

Généralement pas sur WINDOWS mais sur serveur UNIX et encore moins sur son poste mais sur un seveur distant.

Pour les besoins de l'exercice, nous allons déploiier sur son poste WINDOWS.

4.1 Construction et déploiement des jobs

Étape 0 : Cliquer droit sur le job DL_DATA 0.1 et cliquer sur **Construire le job**

```
##########
Dossier d'archive : C:\workspace_talend\build\DL_DATA_0.1.zip
Version 0.1
Tout les options
Scripts DEV
##########
```

Puis dézipper l'archive dans le dossier `C:\workspace_talend\build\DL_DATA`

Étape 1 : Cliquer droit sur le job ALIM_BDD 0.2 et cliquer sur **Construire le job**

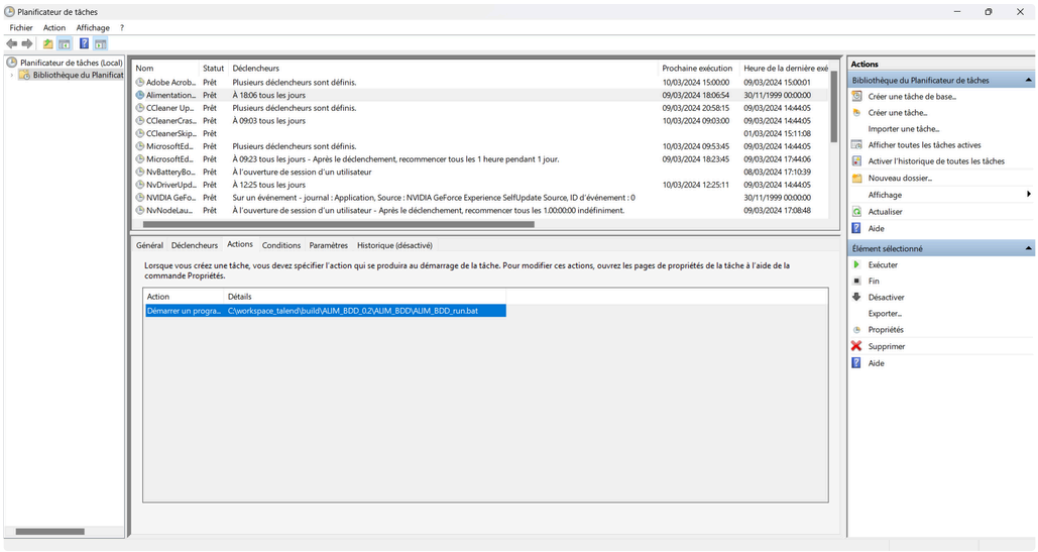
```
##########
Dossier d'archive : C:\workspace_talend\build\ALIM_BDD_0.2.zip
Version 0.2
Tout les options
Scripts DEV
##########
```

Puis dézipper l'archive dans le dossier `C:\workspace_talend\build\ALIM_BDD`

Étape 2 : Il faut suiffit pour le déploiement de programmer les commandes suivantes

```
C:\workspace_talend\build\DL_DATA_0.1\DL_DATA>DL_DATA_run.bat
C:\workspace_talend\build\ALIM_BDD_0.2\DL_DATA>ALIM_BDD_run.bat
```

Par exemple avec un planificateur de tâche :



4.2 Construction et déploiement du service

> Apache Karaf est un conteneur léger et polyvalent basé sur la plate-forme Java. Il est conçu pour l'exécution d'applications et de services s'appuyant sur des composants modulaires.

Étape 0 : Cliquer droit sur le job GET_INFO_BDD 0.1 et cliquer sur **Construire le job**

```
##########
Dossier d’archive : C:\ESB_731\Runtime_ESBSE\container\deploy\GET_INFO_BDD.jar
Type de construction : Bundle OSGi pour ESB
##########
```

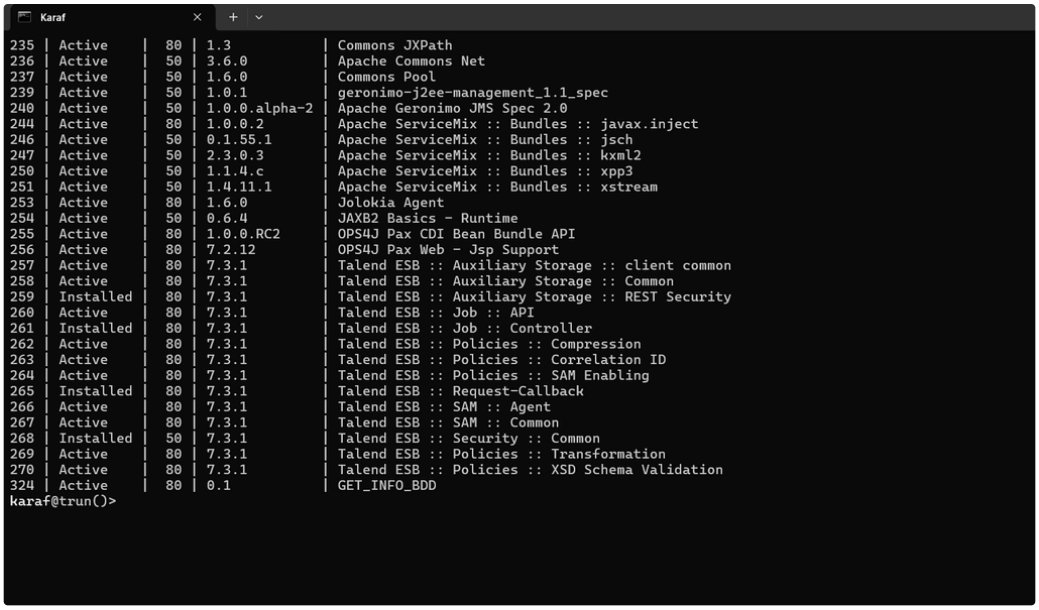
Étape 1 : Se mettre dans le dossier *C :\ESB_731\Runtime_ESBSE\container\bin*

Étape 2 : Lancer la commande

```
trun.bat
```

Étape 3 : Observer qu’un service GET_INFO_BDD est bien déployé, avec la commande dans karaf

```
bundle:list
```



4.2 Construction et déploiement de la route

Étape 0 : Cliquer droit sur la route poste_vote 0.1 et cliquer sur **Construire le job**

```
##########
Dossier d’archive : C:\ESB_731\Runtime_ESBSE\container\deploy\post_vote_0.1.kar
Type de construction : ESB Runtime Kar File
##########
```

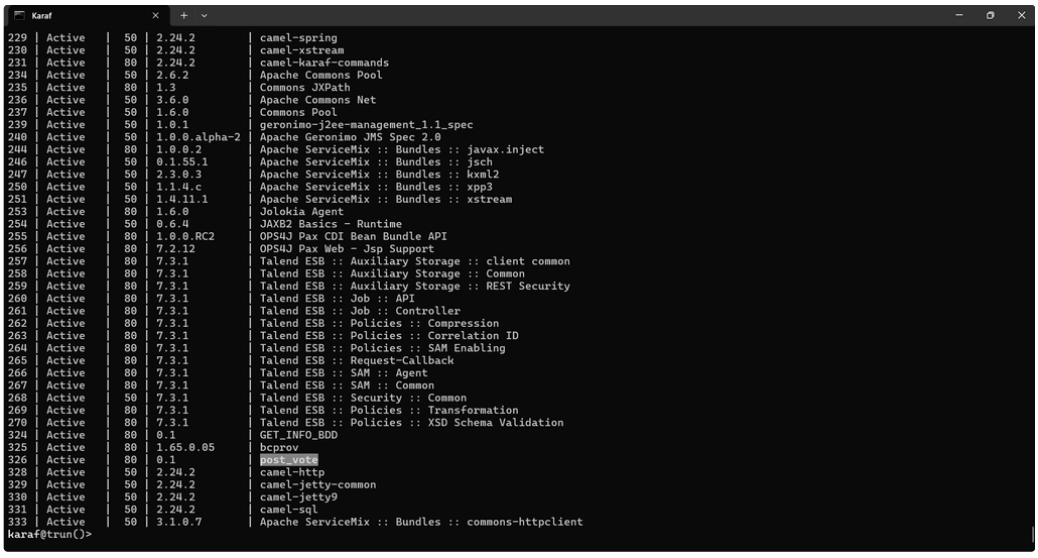
Étape 1 : Se mettre dans le dossier *C:\ESB_731\Runtime_ESBSE\container\bin*

Étape 2 : Lancer la commande

```
trun.bat
```

Étape 3 : Observer qu’un service post_vote est bien déployé, avec la commande

```
bundle:list
```



5 Développement collaboratif avec Git

Il est bien connu que la version Open Source de Talend, Talend Open Studio est mono-poste, c'est-à-dire qu'elle ne permet pas le travail collaboratif intégré à l'outil. Pour cela, il faut se diriger vers les solutions entreprises.

Pourtant, avec un minimum d'esprit de développeur, il est possible de mettre manuellement les sources Talend sur Git.

En effet, les jobs,métadonnées, contextes crées dans Talend sont stockés sous forme de fichiers dans un répertoire spécifique, appelé workspace (cette terminologie parlera bien sûr aux développeurs habitués à Eclipse).

Tout fichier pouvant être versionné sur Git, rien n'empêche donc une mise sous contrôle Git de notre travail Talend.

5.1 Initialisation d'un projet sous Git

5.1.1 Publication sur Git

Suivez les étapes dans l'ordre :

- Démarrez Talend Open Studio en tant qu'administrateur
- Dans **BitBucket**, créez un projet OPENDATA_ASSEMBLEE_NATIONALE puis un dépôt (repository) à l'intérieur avec un fichier README.md par défaut, et enfin récupérez le lien HTTPS (ou SSH si vous avez paramétré une clé SSH) du dépôt
- Ouvrez le client Git et placez-vous au niveau du workspace
- Configurez le Git avec vos identifiants Bitbucket (dans le cas de l'utilisation d'un lien HTTPS)

```
git config --global user.name "q_drousie"
git config --global user.email quentin.drousie@daka-tec.com
```

- Initialisez le repo Git avec la commande

```
git init
```

- Créez un fichier ***.gitignore*** à la racine du workspace en exécutant la commande

```
vim .gitignore
```

- Ajoutez le contenu suivant à personnaliser selon le nom de votre projet (Ici OPENDATA_ASSEMBLEE_NATIONALE)

```
**/jobInfo.properties
.JETEmitters/
.Java/
.metadata/
OPENDATA_ASSEMBLEE_NATIONALE/code/routines/system/*.*
OPENDATA_ASSEMBLEE_NATIONALE/temp/
OPENDATA_ASSEMBLEE_NATIONALE/sqlPatterns/
OPENDATA_ASSEMBLEE_NATIONALE/talend.project
```

- Ajoutez le lien (HTTPS ou SSH, voir plus haut) au dépôt

```
git remote add origin https://q_drousie@bitbucket.org/dakatec
```

Mettez à jour vos sources avec la commande

```
git pull origin master
```

Ajoutez vos modifications en cache avec la commande

```
git add .
```

- Committez vos modifications avec un commentaire avec la commande

```
git commit -m "Votre commentaire"
```

- Publiez les modifications sur Git avec la commande

```
git push origin master
```

- Sur BitBucket, dans le projet talend et le dépôt talend se trouve désormais un dossier OPENDATA_ASSEMBLEE_NATIONALE (nom technique du projet Talend)

5.1.2 Récupération en local d'un projet versionné sur Git

Suivez les étapes dans l'ordre :

- Créez un workspace sur votre poste, à l'emplacement de votre choix

```
C:\workspace_talend
```

- Ouvrez le client Git et placez-vous au niveau du répertoire nouvellement créé

```
cd "C:\workspace_talend"
```

- Configurez le Git avec vos identifiants Bitbucket (dans le cas de l'utilisation d'un lien HTTPS)

```
git config --global user.name "q_drousie"
git config --global user.email quentin.drousie@daka-tec.com
```

Sur BitBucket, récupérez le lien HTTPS (ou SSH si vous avez paramétré une clé SSH) du dépôt - Dans mon exemple, le lien HTTPS du dépôt est

```
https://q_drousie@bitbucket.org/dakatec/talend.git
```

- Clonez le dépôt dans le dossier workspace

```
git clone https://q_drousie@bitbucket.org/dakatec/talend.git
```

- Vérifiez que le dépôt a bien été rapatrié

```
C:\workspace_talend
```

- Démarrez Talend Open Studio en tant qu'administrateur
- Changez le workspace au démarrage pour pointer vers le workspace que vous avez créé et alimenté dans les étapes précédentes (avec le \talend au bout) et redémarrez Talend Open Studio
- Créez un nouveau projet portant exactement le même nom que celui publié sur le Git (dossier se trouvant à la racine du dépôt)

5.2 Cycle de vie d projet

5.2.1 Récupération en local de la dernière version d'un projet Git

Si vous avez déjà récupéré le projet en local et que vous souhaitez de nouveau développer dessus, il faut impérativement que vous mettiez à jour vos sources au préalable.

Ouvrez le client Git et exécutez les commandes

```
git config --global user.name "q_drousie"
```

```
git config --global user.email quentin.drousie@daka-tec.com
cd "Votre workspace"
git remote add origin url_distante_de_votre_projet
git pull origin master
```

5.2.2 Publication sur le Gitde la dernière version d'un projet en local

Une fois la dernière version du projet Git récupéré en local et les développements effectués, poussez votre travail en retour sur le Git.

Ouvrez le client Git et exécutez les commandes

```
git config --global user.name "q_drousie"
git config --global user.email quentin.drousie@daka-tec.com
cd "Votre workspace"
git add .
git commit -m "Résumé de votre travail"
git remote add origin url_distante_de_votre_projet
git push origin master
```

6. Annexe

6.1 Tableau de conversion de type

Type origine	Type cible	Fonction
String	Integer	Integer.parseInt(row1.myString)
String	Integer	(new Integer(row1.myString)).toString()
String	Date	TalendDate.parseDate("dd-MMyyyy",row1.myString)
String	BigDecimal	new BigDecimal(row1.myString) where myString can include decimal places. For example, 99.00
String	Float	Float.parseFloat(row1.myString)
String	Long	Long.parseLong(row1.myString)
Long	String	row1.myLong.toString()
Integer	String	variable+"" or variable.toString()
Integer	Long	row1.myInteger.longValue()
Integer	BigDecimal	new BigDecimal(row1.myInteger)
Integer	Float	new Float(row1.myInteger)
Float	String	row1.myFloat.toString()
Float	Integer	To do this conversion you need to decide on arounding methods such as Math.round(),Math.ceil(),

		Math.floor() and then cast the result to Integer.
Float	BigDecimal	new BigDecimal(Float.toString(row1.myFloat))
Date	String	TalendDate.formatDate(“yy-MM-dd”,row1.myDate)
BigDecimal	String	row1.myBigDecimal.toString()
BigDecimal	Integer	As with Float, BigDecimal can have decimal places, so will need to be rounded prior to casting toInteger.

6.2 Talend et quelques notions Java

== : Égal
!= : Différent
= : Déclaration ou valorisation d'une variable en Java

Xxxx == 100 ou xxxx > 100 ou xxxx < 100 : Test sur la valeur d'un champ numérique

---

"toto".equals("xxx") / !"toto".equals("xxx") : Test d'égalité / inégalité d'un champ de type String

---

Xxxx == null ou xxxx != null
Relational.isNull(Xxxx) / ! Relational.isNull(Xxxx) : Test de la nullité ou non d'un champ

---

Xxxx.isEmpty() / ! Xxxx.isEmpty() : Teste si le champ est vide ou pas (vide <> null : un champ peut être non null mais vide)

Xxxx.startsWith ("xx") / Xxxx.endswith ("xx") : Commencer / Terminer par xxxx, LIKE en SQL ("xx%")/SQL ("%xx")

Xxxx.contains ("xx") : Contenir xxxx , LIKE en SQL ("%xx%")

Boolean : True/ False (si la valeur est en string) ; 1/0 (si la valeur est en int)

6.3 Exemples de conversion Talend

Integer.valueOf("xxxx") : Conversion d'un string en int

String.valueOf(xxxx) : Conversion d'un int en string

---

variable.toString() ou xxxx + "" : Conversion d'un int ou float en string

---

Float.parseFloat("xxxx") : Conversion d'un char en float

BigDecimal("xxxx" ou xxxx) : Conversion en BigDecimal (de string ou integer)

TalendDate.parseDate("dd/MM/yyyy","01/01/2020") : Conversion d'un string en date en précisant le format du string

---

TalendDate.formatDate("dd", XXX) : Afficher uniquement le jour; Il faut que XXX soit au format Date (via parseDate si besoin). La sortie est au format string.

StringHandling.LEFT(xxxx,y) / StringHandling.RIGHT(xxxx,y) : Récupération des Y caractères de la chaîne en partant de la gauche ou de la droite. Cette fonction propre à Talend protège des null.

---

xxxx.substring(0,3) : Du premier au troisième caractère d'une chaîne.

Arrays.asList("X","Y").contains(xxxx) : Champ xxxx dans la liste. Nécessite un « import java.util.Arrays; » dans Advanced settings d'un tJavaRow, par exemple.

StringHandling.CHANGE(chaine ,"caractère à remplacer", "caractère remplaçant")) : Substitution d'un caractère dans une chaîne. Cette fonction propre à Talend protège des null.

Math.abs(xxx) : Valeur absolue version Java (+ fiable), Talend retourne un Double dans tous les cas (10.0).

6.4 Talend et les tests ternaires

Xxxx != null ? Xxxx : null : (Test) ? Valeur si vrai : Valeur si faux

---

Xxxx != 0 ? Xxxx : 0 : En Java, si l'objet est en int, le null n'est pas accepté, il faut mettre un 0.

"Toto".equals(xxxx) ? Xxxx : "XXXX" : Si XXXX = "Toto", alors afficher XXXX, sinon vide

---

("Toto".equals(xxxx)|| "Toto".equals(yyyy)) : **"**XXXXYYYY**"** Si XXXX = "Toto" OU YYYY= "Toto" , alors afficher **XXXXYYYY**, sinon vide

6.5 Les expressions régulières (Regex) dans Talend

Ces expressions servent à définir des modèles pour la recherche et la manipulation de et dans des strings.

Exemple pour enlever les caractères d'un string :

```
"12zer45aaz43".replaceAll("\\D+","");
Résultat : 124543
```

6.6 Les composants Talend les plus utilisés

- **tMap** : C'est le composant le plus important et le plus puissant de Talend. Il permet de réaliser les multi opérations (jointures, transformations, filtres, rejets...) Les expressions utilisées sont en Java.

*NB : **IF THEN ELSE** n'est pas autorisé !*

Le nombre d'entrées et sorties n'est pas limité officiellement, mais pour la maintenabilité, il est fortement conseillé de limiter les entrées à un ou deux, les sorties peuvent être nombreuses.

- **tAgregateRow** : Ce composant reçoit un flux de données et fait une agrégation basée sur une ou plusieurs colonnes. Il permet d'établir des métriques et des statistiques basées sur des valeurs ou des calculs.

NB : Il faut faire attention à ce que l'entrée et à la sortie de la colonne soient identiques. Ce composant met par défaut la première colonne de sortie pour toutes les autres.

Par rapport à SQL, l'ordre est à l'inverse : d'abord Group By ensuite opérations.

Opérations : Min, max, somme, moyenne, compter, premier, dernier, liste...

- **tSortRow** : Ce composant trie les données dans un flux. Pour la performance du traitement, il est conseillé d'utiliser ce composant avant et après une agrégation.
- **tFilterRow** : Il filtre les données dans un flux. Dans la partie "avancé", une expression Java peut être utilisée.

NB : La méthode simple et la méthode avancée ne peuvent pas être cumulées.

- **tFlowToIterate** : Ce composant ne peut être utilisé en début de chaîne. Il permet la transformation d'une liste de valeurs en liste d'exécution. Il effectue une itération sur les données d'entrée et génère des variables globales. Il est utilisé afin de lire des données ligne par ligne.

Dans le paramètre d'iterate il est possible de choisir le nombre d'exécutions en parallèle, c'est de l'**exécution concurrentielle**, ce qui permet que le traitement soit rapide. Dans ce cas, l'exécution est aléatoire. Si l'ordre est important pour le traitement, ne pas utiliser cette méthode.

- **tIterateToFlow** : Ce composant permet de transformer des données non traitables en flux traitable. Ce composant ne peut être utilisé en début de chaîne.
- **tFileList** : Ce composant liste les fichiers d'un répertoire donné (possibilité de masque de fichier).

La méthode à observer est la **récursivité** (inclure les sous-répertoires). Ce composant est capable d'aller lire les fichiers recherchés dans tous les sous-dossiers et de les exécuter en même temps. C'est le **parallélisme**.

Il possède des variables globales tels que :

- CURRENT_FILE : nom du fichier courant
- CURRENT_FILEPATH : nom du fichier courant ainsi que son chemin d'accès
- CURRENT_FILEEXTENSION : extension du fichier courant
- CURRENT_FILEDIRECTORY : répertoire du fichier courant
- NB_FILE : nombre de fichiers itérés
- **tLogRow** : (Affichage par ligne) : C'est un excellent outil de débogage. Il envoie les données vers la console.
- **tNote ou Note** : Il permet d'ajouter des commentaires, une présentation, liste les évolutions du job.

*NB : Le composant est "Note" mais s'appelle en tapant "**tNote**".*

- **tMemorizeRow** : Ce composant mémorise une ou plusieurs lignes et permet au(x) composant(s) suivant(s) d'effectuer des opérations de votre choix sur les lignes mémorisées. Le nombre de lignes à mémoriser doit être mis à 2 ou plus.

*NB : En amont de ce composant, faire un tri en utilisant le composant **tSortRow***

- **tAggregateSortedRow** : Ce composant reçoit un flux de données triées sur lequel il effectue une agrégation basée sur une ou plusieurs colonnes.

Si les données sont déjà triées, les performances n'en sont que plus optimisées.

- **tFixedFlowInput** : Il permet de créer une mini table (colonne + lignes) avec des valeurs en dur.
- **Fichiers plats** : Talend lit et écrit différents types de fichiers plats tels qu'Excel, CSV, positionnel, xml… Chaque type de fichier possède deux composants spécifiques (pour lire « Input », pour écrire « Output »).
- **BDD** : Talend lit et écrit les différents types de BDD tel que MSSQL, MYSQL, ORACLE. Les composants spécifiques principaux sont la connexion à la base de donnée, la lecture, l'écriture et le commit / roll back.

6.7 Talend et les différences entre tJava, tJavaRow et tJavaFlex

Ces 3 composants permettent l'insertion de codes Java personnalisés.

Un composant Java comprend 3 parties : **Begin**, **Main** et **End**.

- **tJava** : Le tJava n'a qu'un Begin donc il ne s'exécute qu'une fois. Il ne gère pas de flux de données (de type row) donc pas de sortie. Rattaché à un job ou à un sous-job par un trigger (onComponentOk ou onSubjobOk)
- **tJava Row** : Le tJava Row n'a que la partie Main.
- **tJavaFlex** : Le tJavaFlex est similaire au tJava. La différence est qu'il crée automatiquement un flux de données en sortie à partir du flux de données en entrée (donc pas besoin d'initialisation). Ce composant a le begin, main et end. Il peut donc être mis seul ou en début de job, ou même au milieu en prenant en compte le flux de données. C'est le plus flexible des trois.

6.8 Format de date

	Locale fr_FR : français, France
Modèles de dates et d'heures	Exemple
22/03/99	22/03/99
d MMM yyyy	22 mars 1999
d MMMM yyyy	22 mars 1999
EEEE d MMMM yyyy	lundi 22 mars 1999
dd/MM/yy HH:mm	22/03/99 05:06
MM/dd/yy HH:mm	03/22/99 05:06
M/d/yy HH:mm	3/22/99 05:06
MM-dd-yy HH:mm	03-22-99 05:06
M-d-yy HH:mm	3-22-99 05:06
d MMM yyyy HH:mm:ss	22 mars 1999 05:06:07
d MMMM yyyy HH:mm:ss z	22 mars 1999 05:06:07 CET
MM-dd-yyyy HH:mm:ss	03-22-1999 05:06:07
M-d-yyyy HH:mm:ss	3-22-1999 05:06:07
yyyy-M-d HH:mm:ss	1999-3-22 05:06:07
dd/MM/yyyy HH:mm:ss	22/03/1999 05:06:07

d/M/yyyy HH:mm:ss	22/3/1999 05:06:07
M/d/yyyy HH:mm:ss	03/22/1999 05:06:07
M/d/yyyy HH:mm:ss	3/22/1999 05:06:07
EEEE d MMMM yyyy HH' h 'mm z	lundi 22 mars 1999 05 h 06 CET
dd/MM/yy HH:mm:ss	22/03/99 05:06:07
MM/dd/yy HH:mm:ss	03/22/99 05:06:07
M/d/yy HH:mm:ss	3/22/99 05:06:07
dd/MM/yyyy HH:mm	22/03/1999 05:06
d/M/yyyy HH:mm	22/3/1999 05:06
MM/dd/yyyy HH:mm	03/22/1999 05:06
M/d/yyyy HH:mm	3/22/1999 05:06
MM-dd-yy HH:mm:ss	03-22-99 05:06:07
M-d-yy HH:mm:ss	3-22-99 05:06:07
MM-dd-yyyy HH:mm	03-22-1999 05:06
M-d-yyyy HH:mm	3-22-1999 05:06
yyyy-M-d HH:mm	1999-3-22 05:06

6.9 Les messages d'erreurs fréquents dans Talend

Data Truncation : La longueur des données d'un champ dépasse sa taille. En général, Talend indique le champ concerné, mais pas dans quelle ligne.

java.lang.NullPointerException : Cela se produit généralement lors de l'utilisation d'un test ternaire dans un tMap et que le cas de nullité de l'objet en entrée n'a pas été traité.

java.lang.NumberFormatException:null :

- Si conversion en nombre d'une chaîne de caractères qui ne représentent pas un nombre.
- Si l'objet est null : remplacer le null par une valeur par défaut (0).
- Si la valeur est divisée par 0.

For input string :

- Si le format ne correspond pas (int dans string par exemple).
- Si l'en-tête du fichier intégré est traité comme une ligne. (nb ligne entête)
- Si le float (12.3) est écrit avec (12,3) : La virgule n'est pas acceptée.

NB : Cette erreur n'empêche pas le traitement, il saute la ligne qui pose problème et continue son traitement.

7. Source

Talend | Une solution de gestion de données complète et évolutive

TP1 - Services Web REST et SOAP avec Talend - TP eServices

List of date and date/time formats | Talend Cloud Data Preparation User Guide Help

- Les bonnes pratiques Talend
- Opendata - Assemblée nationale
- Documentation
- Apache Karaf - The Modulith Runtime
- Utiliser Talend/Karaf pour déployer web services et routes - Talend
- ActiveMQ
- 7.4 ActiveMQ
- FORMATION_TALEND