## Flux comptables boutiques

### Contexte

L’import des fichiers Excel en xls des livrets comptables des boutiques vers X3 requiert l’agrégation de plusieurs données dont la somme des débit et crédit par type d’opération par jours par boutique, afin de pouvoir les intégrer en lignes analytiques gérées par X3. L’automatisation de formatage de ces données permettra un gain de temps conséquent pour leur gestion. Cette automatisation s’effectue via l’IDE Visual Studio 2019 et grâce à l’extension SQL Server Integration Services Projets dans sa version 3.15 installable depuis le gestionnaire d’Extension de l’IDE.

### Environnement de Travail

#### Fonctionnement global

Le fichier de livret comptable doit être déposé dans un dossier de dépôt prévu à cette intention. Ce fichier doit être sous format xls et contenir le codeBoutique de X3 pour les trois 1ers caractères de nom de fichier. Le package sous SSIS va récupérer ce fichier afin de le traiter. Dans un premier temps, il envoie les données du fichier xls vers la base de données Datawarehouse dans la table **dbo.pos\_account\_book**. Puis depuis ce fichier xls, les différentes dates contenues dedans sont récupérées dans une liste afin de pouvoir itérer sur chaque date dans le fichier, permettant ainsi un traitement par date et non pas par fichier. Cette configuration s’avère utile lorsqu’un même fichier xls contient plusieurs dates à traiter.

Ensuite, l’ensemble des données de la table **dbo.pos\_account\_book** sont récupérées selon la boutique concernée et la date. Puis des fonctions d’agrégations par boutique, par date, par numéro de compte, permet de mettre en place la somme des crédits et débits par compte. Ces informations rassemblées, il est alors possible de mettre en place un fichier sous forme csv qui sera automatiquement déposer dans un dossier de dépôt pour X3. Ce dernier sera checké puis importer si celui-ci et valide.

#### Variables du packages

Pour les besoins de ce package, un ensemble de plusieurs variables ont été mis en place.

Une image contenant table

Description générée automatiquement

##### Les variables des chemins

Il existe 5 variables définissant les chemins d’accès **typés en String** vers les dossiers nécessaires au traitement :

* **nameFile** : Contient le nom et l’extension du fichier en cours d’utilisation ou la valeur « tmp.xls »

|  |
| --- |
| @[User::nameFile] == "" ? "tmp.xls" : @[User::nameFile] |

* **pathFolderDepot** : Définit le chemin vers l’adresse où il faut déposer les fichiers xls à traiter

|  |
| --- |
| "C:\\Users\\"+ REPLACE(@[System::UserName], "HYTPROD\\", "" ) +"\\source\\repos\\ETL\_CA\_Tests\\ssis\_CA\_chantier\\depot\\" |

* **pathFolderTemplate** : Définit le chemin d’accès aux fichiers servant de modèles et de base neutres pour les sources.

|  |
| --- |
| "C:\\Users\\"+ REPLACE(@[System::UserName], "HYTPROD\\", "" )+"\\source\\repos\\ETL\_CA\_Tests\\template\\" |

* **pathNameFile**: Définit le chemin d’accès au fichier en cours d’utilisation, son expression est la concaténation des variables de **pathFolderDepot** avec **nameFile**

|  |
| --- |
| @[User::pathFolderDepot] + @[User::nameFile] |

* **pathFolderApres** : Définit le chemin d’accès vers le dossier de dépôt des fichiers en csv pour X3

|  |
| --- |
| "C:\\Users\\"+ REPLACE(@[System::UserName], "HYTPROD\\", "" ) +"\\source\\repos\\ETL\_CA\_Tests\\ssis\_CA\_chantier\\apres\\" |

* **pathFolderArchive**: Définit le chemin d’accès vers le dossier d’Archive des fichiers traités.

|  |
| --- |
| "C:\\Users\\" + REPLACE(@[System::UserName], "HYTPROD\\", "" ) + "\\source\\repos\\ETL\_CA\_Tests\\ssis\_CA\_chantier\\Archives\\" |

##### Les variables d’itérations et de paramétrages

Afin de savoir sur quelle boutique le fichier en cours de traitement fait référence et à quelles dates, des variables ont été mises en place :

* **lstDate** : Elle récupère sous forme d’**Object** énumérable, la liste des dates présentes dans le fichier en cours d’utilisation.
* **dtDate** : De type **DateTime**, cette variable sert à itérer sur chaque valeur de la liste **lstDate**.
* **strDate** : De type **String**, cette variable formate la valeur de **dtDate** en dd/MM/YYY

|  |
| --- |
| MONTH( @[User::dtDate] ) < 10 && DAY( @[User::dtDate] ) < 10 ?  "0"+(DT\_WSTR, 2) DAY( @[User::dtDate] ) +"/0"+  (DT\_WSTR, 2) MONTH( @[User::dtDate] )+"/"+  (DT\_WSTR, 4) YEAR( @[User::dtDate] ) + ""  : MONTH( @[User::dtDate] ) < 10 ?  (DT\_WSTR, 2) DAY( @[User::dtDate] ) +"/0"+  (DT\_WSTR, 2) MONTH( @[User::dtDate] )+"/"+  (DT\_WSTR, 4) YEAR( @[User::dtDate] )  : DAY( @[User::dtDate] ) < 10 ?  "0"+(DT\_WSTR, 2) DAY( @[User::dtDate] ) +"/"+  (DT\_WSTR, 2) MONTH( @[User::dtDate] )+"/"+  (DT\_WSTR, 4) YEAR( @[User::dtDate] ) + ""  :(DT\_WSTR, 2) DAY( @[User::dtDate] ) +"/"+  (DT\_WSTR, 2) MONTH( @[User::dtDate] )+"/"+  (DT\_WSTR, 4) YEAR( @[User::dtDate] ) + "" |

* **shop**: De type String, elle récupère les 3 premiers caractères de **nameFile** valorisé avec nom du fichier et extension en cours d’utilisation valorisé. Ces 3 caractères doivent correspondre au codeBoutique X3.

|  |
| --- |
| SUBSTRING( @[User::nameFile] , 1, 3) |

##### Les variables de connexion string

Dans le **Flux de données**, les sources utilisées doivent nécessairement être reliées à un fichier ou avoir un nom de connexion de source qui existe à chaque instant. Dans les paramètres des sources, il est possible de définir la propriété **connexionString** dans l’onglet Expression avec une variable permettant de définir un nom de source par défaut. Ces chaines de connexion dépendent donc du nom du fichier en cours d’utilisation défini.

* **nameFile** : Contient le nom et l’extension du fichier en cours d’utilisation. Néanmoins selon l’état du package s’il est en phase d’initialisation ou en cours d’usage.

|  |
| --- |
| @[User::nameFile] == "" ? "tmp.xls" : @[User::nameFile] |

* **coStrBookAccount :** Cette chaine de connexion vérifie si la valeur de **nameFile** est vide, ou bien initialisée à « tmp.xls ». Si oui, alors la valeur de la **coStrBookAccount**renvoie l’adresse du fichier xls de source par défaut dans le dossier **Templates : tmp\_BookAccount.xls.**

Sinon, il renvoie le chemin du fichier xls en cours d’utilisation.

|  |
| --- |
| @[User::nameFile] == "" || @[User::nameFile] == "tmp.xls" ?  @[User::pathFolderTemplates] + @[$Package::strTemplateTmBookAccountxls]  : @[User::pathNameFile] |

* **coStrFormatX3 :** Cette chaine de connexion vérifie si la valeur de **nameFile** est vide ou initialisée à « tmp.xls ». Si oui, alors la valeur de **coStrFormatX3**renvoie l’adresse du fichier csv servant de source par défaut dans le dossier Template, **FAC\_VTE\_.csv.**

Sinon, il créé le fichier final en csv dans le dossier depot après traitement avec pour nom de fichier la valeur du paramètre strTemplateFormatX3 + le codeBoutiqueX3 + la date relative au fichier sans les séparateur « / ».

Exemple : **FAC\_VTE\_POD\_02112023.csv**

|  |
| --- |
| @[User ::nameFile] == «  » || @[User::nameFile] == "tmp.xls" ?  @[User::pathFolderTemplates] + @[$Package::strTemplateFormatX3] +".csv"  : @[User::pathFolderApres] + @[$Package::strTemplateFormatX3] +@[User::shop]  +"\_"+ REPLACE( @[User::strDate] , "/", "" ) +".csv" |

##### Paramètres supplémentaires

Enfin, des paramètres généraux ont été mis en place afin de pouvoir modifier facilement la valeur des fichiers par défaut

Une image contenant table

Description générée automatiquement

##### Dossier Template

Les chaines de connexion pour les sources et destinations nécessitant un lien de connexion existant par défaut, un dossier a été créé afin de contenir les fichiers attendus. Lorsque le package n’a pas encore initialisé la valeur de nameFile, alors l’ensemble des chaines de connexion renvoi respectivement vers leur fichier source par défaut. Ce dossier contient ainsi trois fichiers :

**FAC\_VTE\_.csv:** Fichier par défaut pour l’écriture des fichiers d’import X3. Son contenu est vide.

**Lstmodele.txt**: Fichier par défaut pour l’écriture des fichiers xls sous forme de fichier texte.

**Tmp\_BookAccount.xsl**: Fichier par défaut pour la récupération des données des fichier xls.

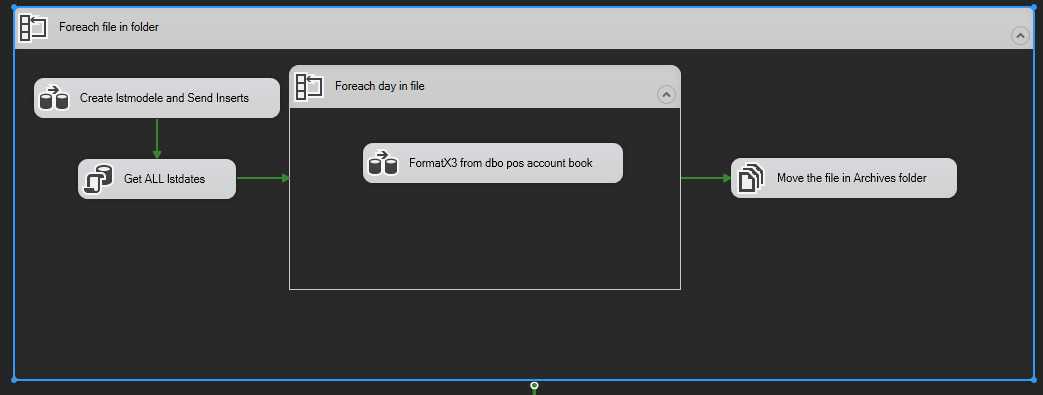
Une image contenant texte

Description générée automatiquement

### Flux de contrôle

#### Conteneur de boucle Foreach file

Dans l’onglet Flux de contrôle, une boucle générale « **Foreach file in folder** » va itérer sur chaque fichier terminant en **.xls** dans le dossier de depot. Aussi, le nom du fichier et son extension est récupéré sous le nom de variables **nameFile.**



Le conteneur est un **conteneur de boucles foreach**, dont l’énumerateur est sur **Enumerateur Foreach file.** Puis, dans l’onglet Mappage de variables, il faut définir une variable qui recevra le nom du fichier et son extension, ici c’est la variable **nameFile.**

Une image contenant texte

Description générée automatiquement

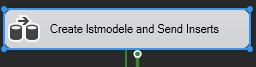
**Il est à noter que la valeur du nom de dossier dans l’énumérateur ne pouvant pas être mise dans une expression, il faut saisir le nom de dossier à gérer par l’éditeur de boucle.**

Une image contenant table

Description générée automatiquement

#### Flux de données Livrets Comptables

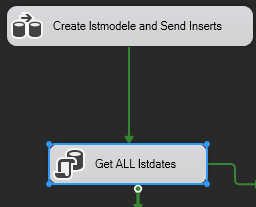
Dans ce conteneur d’itération, un premier **flux de données** est mis en place **Create lstmodele and Send Inserts.** Ce flux permet de récupérer depuis une source Excel en xls, et donc notre fichier déposé, l’ensemble des données contenues dedans afin de les envoyer vers la base de données **Datawarehouse** et dans la table **pos\_account\_book.**



|  |  |
| --- | --- |
| * Date * Piece * N° Compte | * Libelle * Debit * Credit |

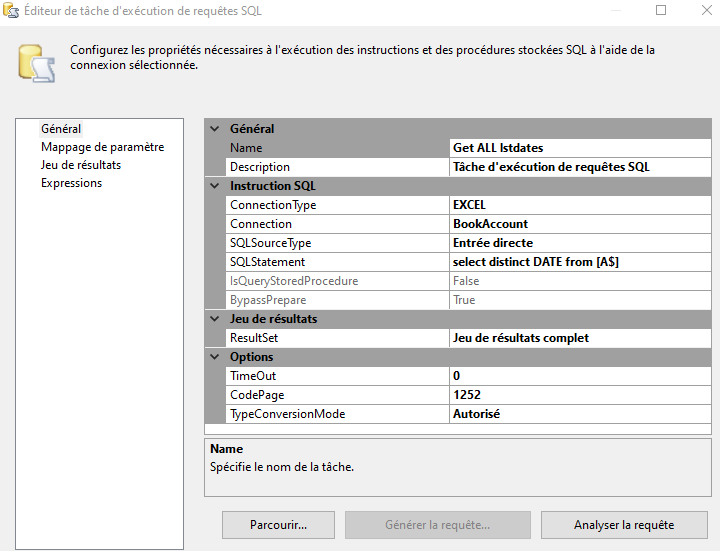
#### Récupération des différentes dates

A la réussite de ce flux de données, une **tâche d’exécution de requêtes SQL** nommée **Get ALL lstdates** est exécutée.



L’exécution de ce dernier demande les configurations suivantes :

* **ConnectionType** : EXCEL
* **Connection** : BookAccount
* **SQLStatement** : Select distinct DATE from [A$]
* **ResultSet** : Jeu de résultat complet



Puis dans l’onglet **Jeu de résultat**, il faut qu’une variable, ici **lstDate** **de type Object** soit placée avec un **nom de résultat** à 0.

Une image contenant table

Description générée automatiquement

#### Conteneur de boucle Foreach day

Suite à cela, un nouveau **conteneur de boucle foreach** nommé **Foreach day in file** est mis en place afin de de pouvoir itérer sur chaque jour que contient le fichier.

Une image contenant texte

Description générée automatiquement

Dans l’onglet **Collection**, le type d’énumérateur utilisé est **un énumérateur ADO Foreach**, la variable source étant **lstDate** qui a été précédemment valorisée par la tâche de requête SQL **Get ALL lstDates.** Puis, pour le mode d’énumération il faut choisir **Lignes dans la première table.**

Une image contenant texte

Description générée automatiquement

Puis dans l’onglet **Mappage de variables,** il faut mettre en place une variable qui servira à itérer chaque valeur dans **lstDate**, ainsi la variable utilisée est **dtDate de type DATETIME** et placer son **index à 0.**

Une image contenant table

Description générée automatiquement

#### Récupération des données depuis pos\_account\_book

Ce conteneur de boucles contient un second **flux de données** nommé **FormatX3 from pos account book.**

Une image contenant texte

Description générée automatiquement

Ce **flux de données** permet de récupérer depuis la table **pos\_account\_book** de la base de données **Datawarehouse,** les informationssuivantes :

|  |  |
| --- | --- |
| * Id * codeBoutique * date * numCompte | * libelle * debit * credit |

Ces données sont par la suite regroupées par nom de boutique, par date, par numéro de compte afin d’en obtenir les sommes de crédits et de débits correspondant. Enfin, un script sous C# 2019 permet de formater les lignes dans le format demandé par X3 pour générer ainsi les fichiers correspondants dans le dossier de dépôt pour X3.

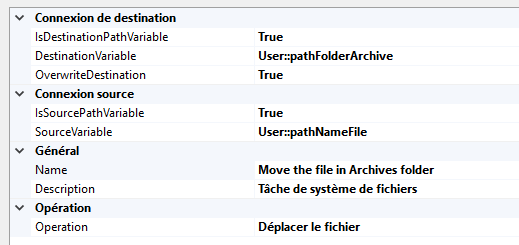
#### Déplacement des fichiers générer

Finalement, une **tâche de système de fichier** nommé **Move the file in Archives Folder**, permet de déplacer le fichier en cours vers un dossier d’Archive de traitement.

Une image contenant texte

Description générée automatiquement

Les configurations mise en place sont les suivantes :



Où **pathFolderArchive** est une variable de **type String** contenant le chemin des archives

**pathFolderNameFile** est une variable de **type String** contenant le chemin du fichier en cours de traitement.

En type d’**opération**, il est nécessaire de choisir **Déplacer le fichier**

### Flux de données

#### Traitement des fichiers sources xls

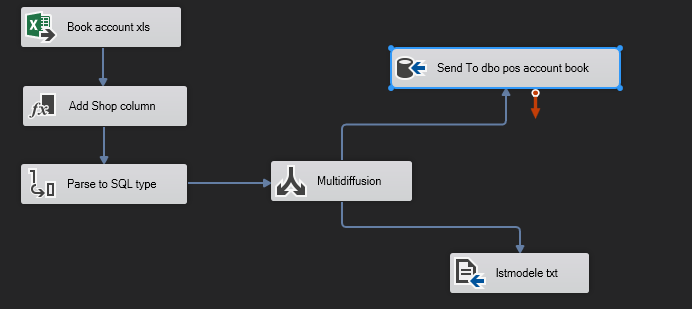
##### Contexte

Une image contenant texte

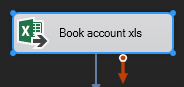
Description générée automatiquement

Situé dans le conteur de boucle foreach file, ce flux de données permet de :

* Récupérer les différentes données du fichier Excel en cours
* Les formater et les convertir dans les formats adéquats
* Les envoyer par insert vers la table pos\_account\_book



##### Source Excel : Book Account xls

Dans l’éditeur de source Excel, le gestionnaire de connexion est réglé sur la connexion **BookAccount** dont la chaine de connexion est lié à la variable **coStrBookAccount**. Afin de récupérer l’intégralité des données une commande SQL demande de sélectionner toutes les données, de toutes les colonnes de la **Feuille A.**

Une image contenant texte

Description générée automatiquement

Ainsi, les données récupérées sont mappées comme il suit

Une image contenant table

Description générée automatiquement

Sous les types correspondants

|  |  |
| --- | --- |
| * DT\_DATE Date * DT\_WSTR Piece * DT\_WSTR N° Compte | * DT\_WSTR Libelle * DT\_R8 Debit * DT\_R8 Credit |

##### Colonne dérivée : Add Shop column

Une image contenant texte, signe, capture d’écran

Description générée automatiquementSuite à la récupération des données, une **colonne dérivée** est mise en place afin de rajouter dans le flux de données une colonne supplémentaire qui précise le **codeboutique** relatif au fichier traité, sa valeur dépend uniquement de celle de la variable de package **shop**. Elle est native de type DT\_WSTR et de taille 3.

Une image contenant table

Description générée automatiquement

##### Conversion de données : Parse to SQL type

Une image contenant texte, signe, capture d’écran

Description générée automatiquement

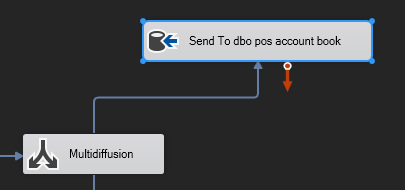
L’ensemble des données récupérées par le fichier Excel sont de type chainé unicode, soit **DT\_WSTR**. Or, afin de pouvoir les intégrer plus facilement dans la table de donnée **pos\_account\_book**, il est nécessaire de les convertir en chaine non unicode, **DT\_STR**. De même, le format de la date depuis excel est de type DT\_DATE, alors qu’il convient de la type pour la base de données.

Ainsi, il est précisé l’ensemble des colonnes d’entrées de type chaines et date avec leur nouveaux nom de sortie.

Une image contenant table

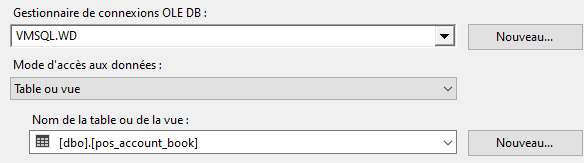
Description générée automatiquement

##### Destination OLD DB : Send to pos\_account\_book

Dans l’éditeur, le gestionnaire de connexion préciser de VMSQL.WD. La connexion à cette source s’effectue dans le **gestionnaire de connexion**, ou bien depuis l’éditeur en cliquant sur « nouveau ». Il suffit alors de préciser :

* **le fournisseur** : OLE DB natif\Microsoft OLE DB Provider for SQL Server
* **Nom du serveur** : VMSQL
* **Sélectionner le nom de base de données** : WD

De retour dans l’éditeur dans de destination OLE DB, il convient d’effectuer les configurations suivantes :



* **Gestionnaire de connexion OLE DB :** VMSQL.WD
* **Mode d’accès :** Table ou vue
* **Nom de la table ou vue :** [dbo].[pos\_account\_book]

Enfin, dans l’onglet **Mappages**, il convient de mapper les colonnes du flux d’entrée avec les données en destination de la base de données :

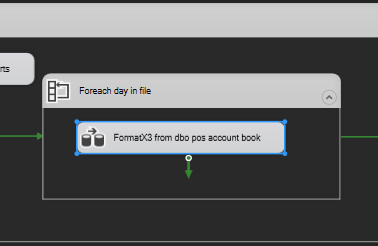
Une image contenant table

Description générée automatiquement

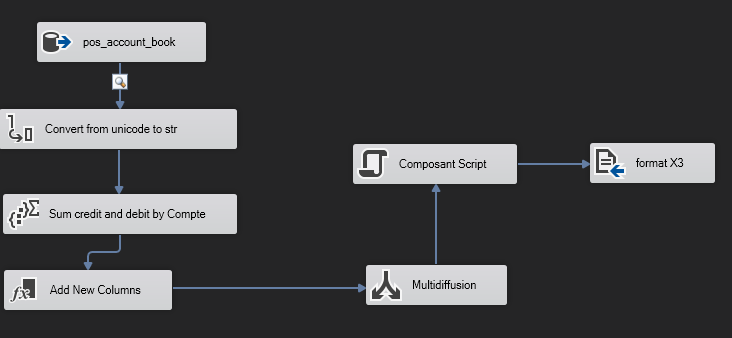
#### Traitement des imports vers X3

##### Contexte

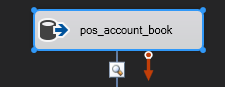
Ce second flux de données s’effectue après récupération de la liste des dates présentes dans le fichier par requête SQL dans le flux de contrôle. Cette liste est ainsi utilisée dans le conteneur de boucle Foreach day in file, où ce conteneur va itérer sur chaque jour présent dans la liste lstDate. C’est alors que le flux de données FormatX3 from dbo pos account book est appelé.



Ce flux contient le schéma suivant :



##### Source OLE DB : pos\_account\_book

Cette source récupère l’ensemble des données depuis la table pos\_account\_book en fonction de la boutique et de la date par l’intermédiaire d’une requête SQL.

Une image contenant texte

Description générée automatiquement

La requête utilisée demande donc deux paramètres, une sur le code boutique et la seconde sur la date. Afin de configurer cela, il convient d’indiquer dans la requête le caractère **‘?’** puis de cliquer sur le bouton des **paramètres**, où la fenêtre suivante apparait.

Une image contenant table

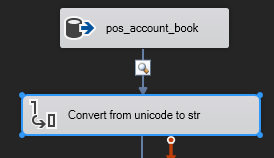
Description générée automatiquement

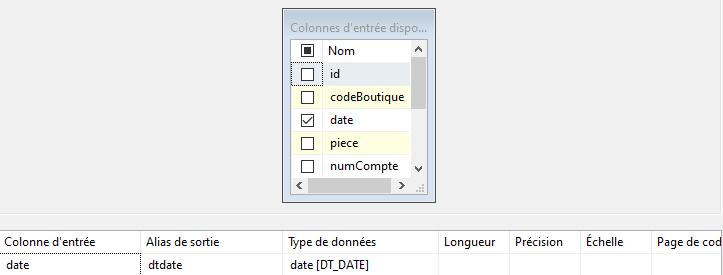
Cette fenêtre permet le mappage des paramètres, il convient alors de les saisir dans l’ordre d’apparition des **‘?’**. Ainsi, le paramètre du champ 0 sera lié à la variable du package **shop** et la seconde à la variable **strDate** qui contient la date concernée formatée. Via la requête SQL, il est alors possible d’obtenir le mappage suivant :

Une image contenant table

Description générée automatiquement

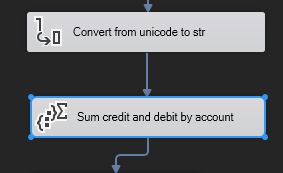
##### Conversion de données : Convert from unicode to str

Le type de la colonne de sortie **date** est une chaine unicode, afin de faciliter son traitement, une opération de conversion est mise en place afin de la convertir en DT\_DATE.



Il est à noter que la nouvelle référence de sortie est maintenant **dtdate** et non plus **date.**

##### Agrégation : Sum credit and debit by account

La transformation agrégation permet d’effectuer des regroupements et des opérations calculs de moyenne, de détermination de valeurs maximales ou minimales ou de sommes.

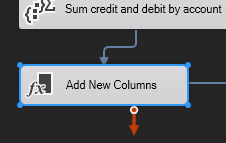
Dans notre cas, le regroupement s’effectue par code de boutique, puis par date, puis par le numéro de compte et libelle. A partir de ce regroupement, il est alors possible de récupérer les sommes des valeurs de crédits et de débits.

Une image contenant table

Description générée automatiquement

Ainsi, il est possible de récupérer en sortie de cette agrégation les lignes distinctes qui nous intéressent puisque une ligne représente désormais une somme de crédit et une somme de débit pour un type de compte, selon une date précise dans une boutique.

##### Colonne dérivée : Add new Colums



Cette transformation permet de créer de nouvelles colonnes de sortie qui seront intégrés au flux de données. Les colonnes ajoutées sont les suivantes :

* montantF : calcule la différence entre la somme des credits et celle des débit.

|  |
| --- |
| sumcredit - sumdebit |

* Id : qui permettra de récupérer l’id de la ligne concernée pour un futur export des fichiers X3 vers datawarehouse.
* Libelle : Permet un pré-formatage du libellé.

|  |
| --- |
| FINDSTRING(libelle,"Vte comptoir",1) == 1 ? "VENTES BOUTIQUES" : UPPER(libelle) |

* **codeBoutique**: Cette variable sert à récupérer la véritable valeur de **shop** attendue via une succession de conditions en ternaire.

|  |
| --- |
| @[User::shop] == "LIP" ? "LHD" : @[User::shop] =="EDM" ? "POD" : @[User::shop] == "PHA" ? "PHA" : @[User::shop] |

##### Composant Script

###### Fonctionnement

Le principe du composant Script dans une flux de données est de boucler sur chaque ligne d’entrée du flux de données, comme un curseur. A chaque ligne, il est alors possible d’effectuer diverses opérations. Aussi, le script met à disposition deux méthodes où il est possible d’agir AVANT la première ligne ***PreExecute()*** et APRES la dernière ligne ***PostExecute()***.

Dans la méthode de boucle sur chaque ligne, ***Entrée0\_ProcessInputRow(), il*** est possible de stocker dans une nouvelle ligne en sortie par appel de deux méthodes :

* Sortie0Buffer.AddRow()
* Sortie0Buffer.OutputName = données à stocker

###### Mise en Garde

Etant donné que la version de l’IDE où le package a été développé est de 2019, sa version du langage utilisé est de 2019. Ainsi, selon la version des IDE où ce package est récupéré il est possible qu’il ait des erreurs, voir une perte du script à son ouverture. Néanmoins, le code du script utilisé ne comportant pas d’imports particuliers qui pourraient engendrer des incompatibilités. Dans ce cas, il suffit de copier les deux codes suivants dans un nouveau composant script avec les configurations qui seront précisés :

Ici, il s’agit du code correspondant au fichier d’ouverture main.cs

|  |
| --- |
| public class ScriptMain : UserComponent  {  public int totalOperation = 0;  /// <summary>  /// This method is called once, before rows begin to be processed in the data flow.  ///  /// You can remove this method if you don't need to do anything here.  /// </summary>  public override void PreExecute()  {    base.PreExecute();  /\*  \* Add your code here  \*/    }  /// <summary>  /// This method is called after all the rows have passed through this component.  ///  /// You can delete this method if you don't need to do anything here.  /// </summary>  public override void PostExecute()  {  base.PostExecute();  /\*  \* Add your code here  \*/  }  /// <summary>  /// This method loop on each row send in the data flow.  /// Then it assign each entries variables with a locale one like this: string shop = Row.strShop  /// Where strShop is a column in the data flow  /// </summary>  /// <param name="Row">The row that is currently passing through the component</param>  public override void Entrée0\_ProcessInputRow(Entrée0Buffer Row)  {  // Links columns by variables  string shop = Row.strShop;  string libelle = Row.strLibelle;  string numCompte = Row.strCompte;  float priceF = (float) Row.montantF;  DateTime dtDate = Row.dtdate;  // This ligne create a objet LigneFormat which is a string with one or 3 lignes.  // The content of LigneFormat depend on the values of strShop, strlibelle, strCompte, montantF, the date and total number of operations;  LigneFormat strRow = new LigneFormat(shop, dtDate, numCompte, libelle, priceF, totalOperation);  // These global value will increase after a row have been processed  if(totalOperation == 0) { totalOperation += 2;}  else { totalOperation += 1; }    // Let to add a new row in the input data flow, then to write the value of LigneFormat in the input column ligneCAX3  Sortie0Buffer.AddRow();  Sortie0Buffer.ligneCAX3 = strRow.ToString();  }  public override void CreateNewOutputRows()  {  /\*  Add rows by calling the AddRow method on the member variable named "<Output Name>Buffer".  For example, call MyOutputBuffer.AddRow() if your output was named "MyOutput".  \*/    }  } |

Ensuite, il faut générer une nouvelle Classe nommée LigneFormat.cs. Cela s’effectue via l’onglet Projet en haut, puis en cliquant sur ‘Ajouter une classe’ et de saisir le nom attendu. Dans l’explorateur de solution

|  |
| --- |
| class LigneFormat  {  // Links columns by variables  string shop;  string libelle;  string nbCpt;  int totalOperation;  DateTime dtDate;  // Variables for builder  string numOp;  string typeOp;  int SensOp = 2;  string strPrice;    public LigneFormat(string shop, DateTime dtDate, string nbCpt, string libelle, float fltPrice, int totalOperation)  {  this.shop = shop;  this.libelle = libelle + " " + dtDate.Day + dtDate.ToString("MM") + dtDate.Year; ;  this.nbCpt = nbCpt;  this.totalOperation = totalOperation;  this.dtDate = dtDate;  // Price is negatif AND Set sens operation  if(fltPrice < 0)  {  fltPrice \*= -1;  if (nbCpt.Contains("707") || nbCpt.Contains("445713") || nbCpt.Contains("445715") || nbCpt.Contains("445716")) { SensOp = 1;}  else{SensOp = 1;}  }  else {  if (nbCpt.Contains("707") || nbCpt.Contains("445713") || nbCpt.Contains("445715") || nbCpt.Contains("445716")) { SensOp = -1;}  else{ SensOp = -1;}  }  strPrice = fltPrice.ToString().Replace(",", ".");  }  override  public string ToString()  {  string str = "";  if (totalOperation <= 0){  str += "G;FAC;;" + shop + ";VTE;" + dtDate.Year + dtDate.ToString("MM") + dtDate.Day + ";"+libelle+";EUR\r\n";  totalOperation += 1;    }  if (nbCpt.Contains("707"))  {  str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + SensOp + ";" + strPrice + ";EUR\r\n";  str += "D;" + totalOperation + ";" + 2 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + SensOp + ";" + strPrice + ";EUR\r\n";  str += "A;" + 1 + ";AX1;1502;AX2;;AX3;" + shop + ";;;;;;;;;;;;;0;"+ strPrice ;  }  else if (nbCpt.Contains("445713") || nbCpt.Contains("445715") || nbCpt.Contains("445716"))  {  str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + SensOp + ";" + strPrice + ";EUR";  }  else if (nbCpt.Contains("531000") || nbCpt.Contains("532000") || nbCpt.Contains("533000") || nbCpt.Contains("580000"))  {  if (nbCpt.Contains("531000") || nbCpt.Contains("532000") || nbCpt.Contains("533000"))  {  numOp = "411000";  typeOp = nbCpt.Contains("531000") ? shop + "ESP" : nbCpt.Contains("532000") ? shop + "CHQ" : nbCpt.Contains("533000") ? shop + "CB" : "!UNK!";    str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";C1;" + numOp + ";" + typeOp + ";" + libelle + ";" + SensOp + ";" + strPrice + ";EUR";  }  else if(nbCpt.Contains("580000")) {  numOp = "467500";  str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + numOp + ";;" + libelle + ";" + 1 + ";" + strPrice + ";EUR";  }  }    else  {  str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + "2" + ";" + strPrice + ";EUR";  }      return str;  }  } |

Enfin, il est à noter qu’en cas d’erreur dans le script, un symbole en forme de croix rouge apparaitra à coté du conteneur de composant Script, indiquant que le script est mal compilé ou qu’il ne trouve pas le code binaire.

Cela peut venir de plusieurs choses :

* Soit il existe une réelle erreur dans le script, alors il convient de rouvrir le script en cliquant sur ‘Modifier le script’ puis de résoudre les éventuelles erreurs puis de le sauvegarder.
* Soit des modifications externes ont impacté les colonnes d’entrées du script, ainsi il suffit de réouvrir le script, de le laisser se charger, puis de le sauvegarder et de le fermer. Le script sera alors à jour.

##### Configuration de l’Editeur

Dans l’onglet Script, il est possible de récupérer des variables de packages en écriture ou en lecture. Ainsi il est nécessaire de les préciser dans les propriétés personnalisées : ReadOnlyVariables ou ReadWriteVariables. Dans notre cas, aucune de nos variables ne sont nécessaires.

Dans l’onglet Colonnes d’entrée, il est nécessaire de sélectionner les colonnes qui serviront, ainsi que leur dénomination dans le script ainsi que de savoir si elles sont en ReadOnly où en ReadWrite. Nous obtenons ainsi la figure suivante :

Une image contenant table

Description générée automatiquement

Enfin, dans l’onglet **Entrées et sorties**, il faut préciser l’ID d’entrée des lignes de la **sortie 0.** Dans **SynchronousInputId** il suffit de le marquer à **Aucun.**

Dans cette **sortie0,** il faut créer une **colonne de sortie**, ici nommée **ligneCAX3**. Son **CodePage** est à 1252 et son DataType sur **Chaîne DT\_STR**.

##### Description du Script

###### Main.cs

Comme précisé, la majorité du script s’effectue dans la méthode Entrée0\_ProcessInputRow() où il reçoit une ligne en paramétre. L’ensemble des colonnes d’entrées sont récupérées ainsi :

String uneVariable = Row.nomAliasColonne

Par exemple :

|  |
| --- |
| string shop = Row.strShop ; |

De cette manière les colonnes de shop libelle, numCompte et montantF sont récupérés dans des variables locales respectives.

Puis, un objet nommé strRow de type LigneFormat est créé à partir de ces variables

|  |
| --- |
| LigneFormat strRow = new LigneFormat(shop, dtDate, numCompte, libelle, priceF, totalOperation); |

**totalOperation** est une variable globale à la classe **ScriptMain**, soit le fichier **main.cs** actuel. Cette variable est initiée et valorisée à 0 en dehors des méthodes, lui permettant ainsi d’être indépendante du processus. Cette variable est incrémentée à chaque nouvelle ligne, puisqu’elle a pour but de compter le nombre d’opération récupéré.

|  |
| --- |
| If (totalOperation == 0) { totalOperation += 2;}  else { totalOperation += 1; } |

Puis la ligne créée par l’objet strRow, est injectée dans la sortie0 de la manière suivante

|  |
| --- |
| Sortie0Buffer.AddRow();  Sortie0Buffer.ligneCAX3 = strRow.ToString(); |

###### Classe LigneFormat

La classe possède un ensemble de variables qui récupère les paramètres envoyés au constructeur, ainsi que quatre variables globales :

* **numOp** : Numéro de compte selon G5
* **typeOp** : S’il s’agit d’une opération en chèque, espèce, carte banque ou carte cadeau
* **SensOp** : Le sens de la transaction
* **strPrice** : Le montant de la transaction

Concernant les variables dans le constructeur

|  |
| --- |
| LigneFormat(string shop, DateTime dtDate, string nbCpt, string libelle, float fltPrice, int totalOperation) |

* **libelle** récupère la valeur issue de sa colonne en entrée à laquelle on ajoute la date du format (ddMMYYY)
* **fltPrice** : il faut vérifier si sa valeur est négative, dans ce cas on multiplie par -1 sa valeur afin de la rendre positive, aussi, on inverse le sens de la transaction. Puis, sa valeur est convertie en chaine afin de pouvoir remplacer sa virgule par un point en tant que séparateur de décimal.

Enfin, une méthode **ToString()** est utilisée en **override** afin de pouvoir la réécrire.

L’objectif étant d’écrire des lignes codifiées, cette méthode contient une succession de condition if/else qui vont valoriser et se concaténer à une variable chainée **str** selon les spécificités précisées dans l’analyse des besoins :

* S’il s’agit de la première ligne d’entrée : Ainsi on ajoute la ligne de type G à str

|  |
| --- |
| str += "G;FAC;;" + shop + ";VTE;" + dtDate.Year + dtDate.ToString("MM") + dtDate.Day + ";"+libelle+";EUR\r\n"; |

* Selon le numero de compte : **nbCpt**
  + S’il contient la valeur ‘707’, on ajoute les lignes D et A dans le format

|  |
| --- |
| str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + SensOp + ";" + strPrice + ";EUR\r\n" ;  str += "D;" + totalOperation + ";" + 2 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + SensOp + ";" + strPrice + ";EUR\r\n";  str += "A;" + 1 + ";AX1;1502;AX2;;AX3;" + shop + ";;;;;;;;;;;;;0;"+ strPrice ; |

* + S’il contient 445713, 445715, 445716

|  |
| --- |
| str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + SensOp + ";" + strPrice + ";EUR"; |

* + S’il contient 531000, 532000, 533000 ou 580000

|  |
| --- |
| numOp = "411000";  typeOp = nbCpt.Contains("531000") ? shop + "ESP" : nbCpt.Contains("532000") ? shop + "CHQ" : nbCpt.Contains("533000") ? shop + "CB" : "!UNK!";    str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";C1;" + numOp + ";" + typeOp + ";" + libelle + ";" + SensOp + ";" + strPrice + ";EUR"; |

* + S’il contient 580000

|  |
| --- |
| numOp = "467500";  str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + numOp + ";;" + libelle + ";" + 1 + ";" + strPrice + ";EUR"; |

* Sinon, si le numéro de compte n’est pas reconnu alors une condition par défaut a été mise en place dans la valeur de l’opération est à ‘2’, ce qui n’est possible.

|  |
| --- |
| str += "D;" + totalOperation + ";" + 1 + ";" + shop + ";;" + nbCpt + ";;" + libelle + ";" + "2" + ";" + strPrice + ";EUR"; |

Finalement, la valeur de **str** est retournée à l’appel de la méthode dans **main.cs,** grâce à la ligne :

|  |
| --- |
| Sortie0Buffer.ligneCAX3 = strRow.ToString(); |

#### Destination de fichier plat : format X3

Dans l’éditeur de destination de fichier plat, le gestionnaire de connexion est établi sur la connexion nommée FormatX3 aussi le mappage s’effectue de ligne à ligne. Dans le gestionnaire de connexion de fichiers plat, il faut préciser que le séparateur de ligne est [CR][CF] et dans l’éditeur de colonne, une seule colonne suffit afin de pouvoir mapper comme il suit :

Une image contenant table

Description générée automatiquement

Enfin, il est à noter que cette connexion dépendant **l’expression** de sa propriété **ConnectionString** définit par la variable de package :**coStrFormatX3**

## Interface vracs

### 4.8.1 Contexte

La mise à jour des données des produits vracs pour les boutiques requiert pour chaque produit de saisir manuellement les données nécessaires, notamment les données de lot, de date limite d’utilisation (dlu), d’origine et de prix. Cette saisie doit aussi s’effectuer pour chaque appareil, de chaque boutique. Elle permet de mettre à jour les données des étiquettes de ventes et d’homogénéiser les données des boutiques et celle des bases de données. Ainsi la saisie manuelle peut être lente et rébarbative étant donnée le nombre de fichier à mettre à jour puis à envoyer sur le site. Afin de contrer ce problème, une automatisation de mise à jour de ces données doit être mise en place. Cette automatisation s’effectue via un package créé sous SSIS (SQL Server Integration Service)

### Préparation environnement de travail

#### Flux de contrôle

Dans l’onglet ‘Flux de contrôle’, plusieurs objets permettent la récupération puis de la transformation des données :

* La tâche de système de fichiers, **‘Create tmpVrac Folder’** permet de créer un dossier temporaire où seront stockées les différents fichiers boutiques.
* La tache d’exécution de processus ‘**Cmd Create product file’** permet, via l’invite de commande windows, de créer un fichier **product.txt** qui servira de base pour la création des fichier produits.
* La tâche d’exécution de requêtes **SQL ‘SQL lstShop’** permet de récupérer la liste des boutiques concernées par les vracs.
* Le conteneur de boucles Foreach **‘Foreach shop in lstShop’** permet de boucler sur chaque boutique.

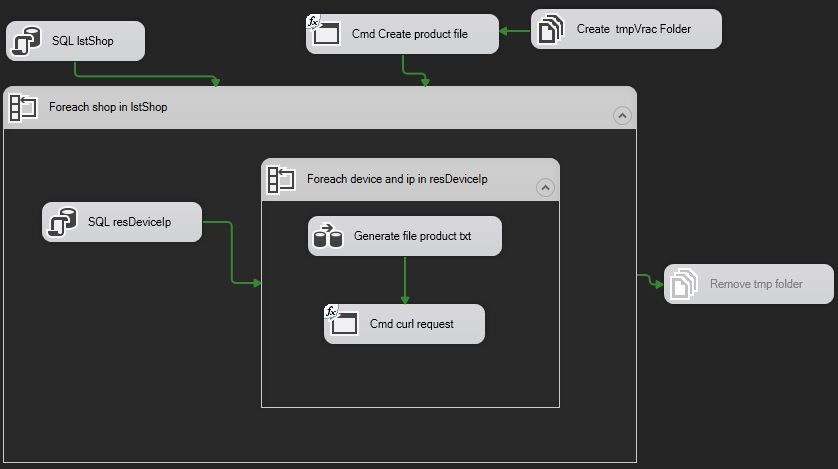
Dans cette boucle, plusieurs opérations ont lieu :

* Une deuxième tache d’exécution de requêtes SQL **‘SQL resDeviceIp’** récupére la liste des appareils vrac et leur ip respectifs
* Un second conteneur de boucles Foreach, **‘Foreach device and ip in resDeviceIp’** permet de boucler selon le nombre de ligne ip/device existant dans la boutique concernée.

Dans cette boucle, on a :

* + Une tâche de flux de données **‘Generate file product txt’** qui permet la récupération des données des produits vracs ainsi que le formatage des données nécessaires, puis créé les fichiers de destinations temporaires.
  + Une tâche d’exécution de processus qui via l’invite de commande créée une requête curl permettant l’envoie des fichiers temporaires vers le web service de Jean bouteille
* Enfin la tache de système de fichiers **‘Remove tmp folder’** permet la suppression du fichier temporaire créé.

Ainsi, le processus obtenu est le suivant :



#### Création des tables de données

Afin d’obtenir l’intégralité des données nécessaires à cette automatisation, plusieurs bases de données ont été mise en place dans la base VMSQL.

* **dbo.pos\_vrac** contient un tableau de correspondance entre le numéro de produit dans les bases ERP, **no**, son équivalent chez Jean Bouteille, **vrac\_no** et celui de la base DIP et RD, **rd\_no**.
* **dbo.pos\_vrac\_support** contient l’ensemble des différents résultats de scan des étiquettes de bidon en boutique. Ces champs sont la boutique concernée, **pos**,l’appareil concerné par la mise à jour produit, **device**, le numéro de produit ERP, **vrac\_no,** le numéro de lot, **lot**, son numéro de sous lot, **slot**, et la dernière date de scan, **scan\_date**.
* **dbo.app\_rd\_dip** contient l’ensemble des données présentes dans les bases RD et DIP nécessaire à la production des étiquettes. Cette base rassemble ainsi le numéro d’article, **no**, la liste des ingrédients du produit, **ingredients**, si la mention de Péremption Après Ouverture (pao) est nécessaire, **pao\_enabled**, sa valeur de pao, **pao**, sa catégorie de vente, **category** ainsi que son **status**
* **dbo.erp.vrac.support** contient l’ensemble des données des étiquettes.
* La table **dbo.products** permet de récupérer le reste des données indispensables à la création des étiquettes formatées telles que le numéro de produits, la description, la dénomination scientifique lorsqu’il est demandé, la catégorie de produit, à savoir s’il s’agit d’une huile végétale, d’une base ou bien d’un savon, son prix au litre, sa densité puis enfin, s’il s’agit d’un produit bio et/ou cosmétique.

#### Création du dossier de destination

**‘Create tmpVrac Folder‘** sert à la création du dossier de destination des différents fichiers boutiques. La variable d’environnement du package **strFolder** permet de recevoir l’adresse texte du dossier.

* **strFolder** : "C:\\tmpVrac\\"

Dans l’éditeur de tache de système de fichiers, les paramètres sont les suivants :

#### 

Puis, **‘Cmd Create product file’** permet de créer le fichier **product.txt** qui contient la chaine «lstbulk;». Il convient de préciser dans la propriété **Executable** le chemin de l’exécutable, ici l’invite de commande. Ainsi sa valeur est : **C:\Windows\System32\cmd.exe**

La propriété **Argument** contient la ligne de commande à exécuter. Cette propriété contient la valeur de la variable de package **tmpVrac** : **"/C echo lstbulk; >product.txt & move product.txt "+ @[User::strFolder] .** Cette propriété peut être soumise à une variable lorsqu’elle est modifiée dans l’onglet **Expression.**

La propriété **WorkingDirectory** doit correspondre au dossier de démarrage de l’executable. Ici fixée sur : **C:\**

Enfin, la propriété **StandardInputDirectory** contient la valeur de la variable **strFolder**, permettant à la variable **tmpVrac** d’être modulable en cas de changement.

Une image contenant texte

Description générée automatiquement

### Récupération liste des boutiques

Le conteneur de boucle Foreach **‘shop in lstShop** permet de récupérer le nom des boutiques concernée par la vente de vracs dans un set de résultats d’une requête SQL.

#### Variables nécessaires

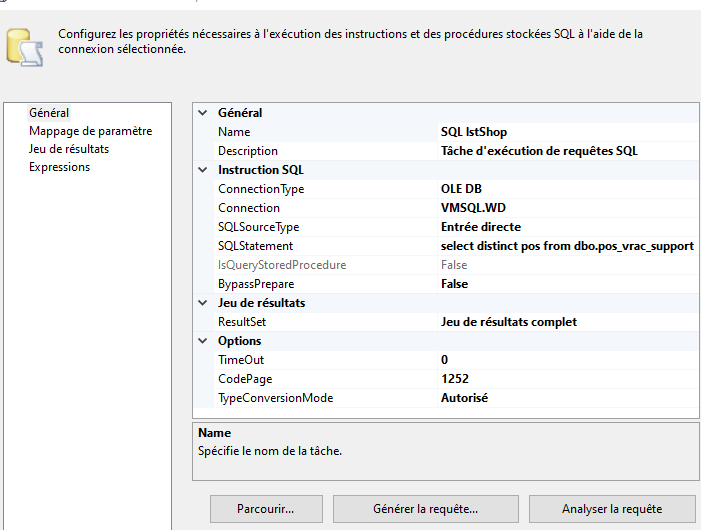
La variable **lstShop** permet de récupérer la liste des différentes boutiques concernées par la vente de produits en vrac, soit Aix en Provence, Odéon, Lille et Metz, dont les dénominations dans les bases VMSQL sont respectivement notées AIX, EDM, LIP, MEP. Elle est de type objet.

De plus, une seconde variable nommée **shop** de type chaine et valorisée à vide doit être créé afin de récupérer les valeurs de **lstShop.**

##### Récupération de la liste des boutiques concernées par le vrac

Via la tâche d’exécution de requêtes SQL, ‘**SQL lstShop’**, la liste des boutiques concernées par la vente de produits vrac va être récupérée, soit Aix en Provence, Odéon, Lille et Metz, dont les dénominations dans les bases de données sont vrac respectivement AIX, EDM, LIP, MEP. Les propriétés de cette tache sont les suivantes :

* Connection Type : OLE DB
* Connection : VMSQL.WD
* SQLSourceType : Entrée directe
* SQLStatement : **Select distinct pos from dbo.pos\_vrac\_support**
* ResultSet : Jeu de résultats complet
* Enfin dans l’onglet ‘Jeu de résultat’, il est précisé dans la colonne ‘ Nom de résultat’ : 0 et en ‘Nom de variable’ : lstShop



#### Boucler dans ‘shop in lstShop’

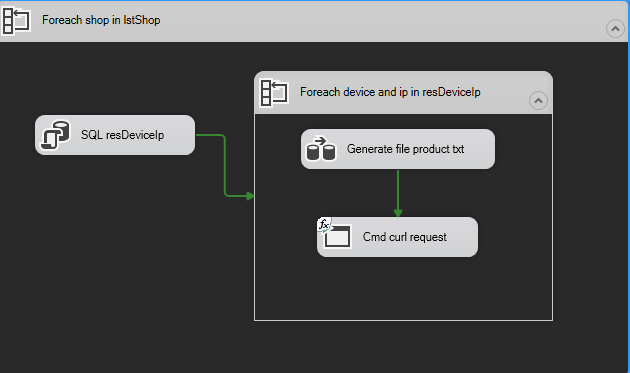
* **Dans l’éditeur de boucle Foreach, il est important de mettre l’Enumerator sur Enumérateur ADO Foreach. De placer lstShop en tant que variable source de l’objet ADO. Puis de cocher ‘Lignes dans la première table’ dans le mode d’énumération.**
* **Dans l’onglet Mappage de variables, dans la colonne Variable, il faut placer la variable shop** à l’index **0.**

**Une image contenant texte

Description générée automatiquement**

### **Récupération des appareils**

Le conteneur de boucle Foreach **‘device and ip in resDeviceIp’** permet de récupérer le nom des appareils vracs et leur ip dans un set de résultats d’une requête SQL en fonction de la boutique **shop** concernée. Ainsi, la même logique est appliquée que pour la récupération de la liste des boutiques. Ainsi, la figure suivante est obtenue :



#### Variables nécessaires

* La variable **shop** est récupérée afin de servir de paramètre dans la requête SQL de récupération des noms d’appareils.
* Deux variables vides et de types chainées, doivent être créés afin de récupérer les valeurs des colonnes de la requête. Ici **Ip** et **device**.
* Une variable objet nommée **resDeviceIp** de type objet qui récupère le résultat de la requête.

#### Liste Nom d’appareils et leur IP

La tache d’exécution de requêtes SQL **‘SQL resDeviceIp’** récupère la liste des appareils ainsi que leur adresse ip respectif pour une boutique **shop** donnée. Dans l’éditeur, les paramètres suivants sont établis :

* Connection Type : OLE DB
* Connection : VMSQL.WD
* SQLSourceType : Entrée directe
* SQLStatement :

|  |
| --- |
| SELECT DISTINCT ip, device  FROM pos\_vrac\_devices  WHERE (pos = ?)  ORDER BY device, ip |

* ResultSet : Jeu de résultats complet
* Enfin dans l’onglet ‘Jeu de résultat’, il est précisé dans la colonne ‘ Nom de résultat’ **: 0** et en ‘Nom de variable’ : **resDeviceIp**

Une image contenant texte

Description générée automatiquement

Aussi il est important de paramétrer dans l’onglet ‘Mappage de paramètre’, la variable **shop** en **Input** afin que la requête puisse s’exécuter correctement.

#### Boucler dans ‘device and ip in resDeviceIp’

* **Dans l’éditeur de boucles Foreach, mettre l’Enumerator sur Enumérateur ADO Foreach. Puis, placer resDeviceIp en tant que variable source de l’objet ADO. Puis de cocher ‘Lignes dans la première table’ dans le mode d’énumération.**
* **Dans l’onglet Mappage de variables, dans la colonne Variable, device et ip** recevront dans la boucle foreach successivement les valeurs des résultats des lignes de **resDeviceIp**. device est à l’**index** est à 1 et ip à l’index 0.

### Récupération et formatage des données vrac

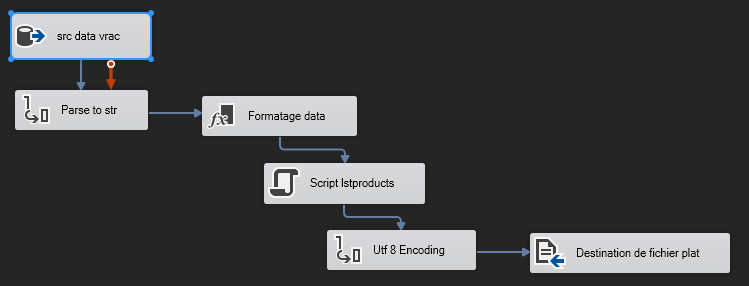
Au sein du conteneur de boucle Foreach pour ‘**device and ip in resDeviceIp**, un flux de données permet de récupérer les diverses données des produits vracs, de générer les fichiers à envoyer. Puis une tâche d’exécution de processus, une requête curl permet de les envoyer. Ainsi, la figure suivante est obtenue :

Une image contenant texte

Description générée automatiquement

#### Récupérer et formater dans Generate All product txt

Dans le flux de données ‘Generate All product txt’, les données des vracs sont récupérées via une source OLE DB et une requête SQL, puis progressivement transformées par des tâches de conversion et un script de formatage avant d’être stockée dans un fichier plat de destination.



##### Récupération ’src data vrac’

La source OLE DB ’src data vrac’ permet de récupérer l’ensemble des données nécessaires au traitement et à la mise à jour des fichiers vracs. On récupère ainsi les flux des données des bases VMSQL.WD. Ici seules les bases de **dbo.products**, **dbo.erp\_vrac\_support,** **dbo.pos\_vrac\_support,** **dbo.pos\_vrac** et **dbo.app\_rd\_dip** nous intéressent ici.

Etant donnée la présence du paramètre **device** dans la requête, le mode d’accès aux données dans l’éditeur de source doit être depuis une commande SQL à partir d’une variable, sans quoi une erreur de violation d’accès bloque le traitement. La variable chainée contenant la requête est **req**, dont le contenu est le suivant :

|  |
| --- |
| "  SELECT pos.no, vrac\_no, description, item\_family, scientific\_description, bio, cosmetic, price\_incl\_tax, density, pao, ingredients, lot, origin, dlu, rd.precautions\_etiquette, rd.mode\_emploi  FROM dbo.products pro LEFT JOIN dbo.pos\_vrac pos ON (pro.no = pos.no)  LEFT JOIN dbo.app\_rd\_dip rd ON (pos.rd\_no = rd.no)  LEFT JOIN (    SELECT SKU, pos.lot, slot, origin, dlu, device, pos.pos, scan\_date  FROM dbo.pos\_vrac\_support pos JOIN dbo.erp\_vrac\_support erp ON (pos.vrac\_no = erp.SKU) AND pos.lot = erp.lot AND pos.slot = erp.slo  JOIN (  SELECT last.pos, last.vrac\_no, MAX(id) AS maxid  FROM dbo.pos\_vrac\_support AS last  GROUP BY last.pos, last.vrac\_no  ) AS req  ON (pos.pos = req.pos) AND pos.vrac\_no = req.vrac\_no AND pos.id = req.maxid    WHERE pos.device = '" + **@[User::device]** +"'  ) AS maj  ON (pro.no = maj.SKU)  WHERE pro.no LIKE '%V'  ORDER by vrac\_no  " |

Les champs récupérés sont les suivants :

* pos.no : numéro du produit dans l’ERP
* vrac\_no : numero de vrac chez JeanBouteille
* description : libellé du produit, renommé name
* item\_family : categorie (2 : huile végétale || 0 : autre ), renommé category
* scientific\_description : Nom INCI
* bio : booléen, vrai si bio
* cosmetic : booléan vrai si produit cosmetique
* price\_incl\_tax : prix au litre, renommé price
* density : densité du produit
* pao : Période après ouverture
* ingredients : liste des ingrédients
* lot : numero de lot
* origin : pays d’origine
* dlu : Date limite d’utilisation
* rd.precautions\_etiquette : liste des précautions d’emploi, renommé etiquette
* rd.mode\_emploi : liste des précautions d’emploi

Il peut être important de noter qu’un champ ‘risk’ contenant les risques éventuels d’une mauvaise utilisation du produit peut être envisagé ; notamment pour l’huile de moutarde qui peut être trop forte pour certaines peaux sensibles. Aussi, les listes des précautions ne concernent que les produits de type Bases et Savons, puisque les Huiles ne sont pas contenues dans la base dbo.app\_rd\_dip et ne possèdent pas précautions particulières précisées dans l’application RD et DIP. Ces informations seront néanmoins ajoutées en brute dans l’attente d’un traitement adéquat. Ainsi dans l’onglet Colonnes, on obtient la figure suivante :

Une image contenant table

Description générée automatiquement

Enfin, il est nécessaire d’ouvrir l’’ ‘**Editeur avancé pour src data vrac** via un clic droit sur la source. Dans l’onglet ‘Propriété d’entrée et de sortie ‘, dans ‘Sortie de source OLE DB’ il est important de modifier la propriété **IsSorted** afin de la valoriser à **true** et de valoriser la propriété **SortKeyPosition** à **1** pour la colonne de sortie **vrac\_no.** Ces modifications permettront d’alléger le traitement et de trier directement la sortie sur le numero de produit vrac.

Une image contenant texte

Description générée automatiquement

Dans ce même onglet, les Colonnes de sorties et d’entrées pour ingredients, mode\_emploi et precaution\_etiquettes doivent avoir leur propriétés **CodePage** à la valeur **65001.** De même, dans l’onglet ‘Propriétés du composant’ la propriété **DefautlCodePage** doit être valorisée à **65001** qui correspond à un encodage en UFT-8.

Une image contenant texte

Description générée automatiquement

##### Conversion en type chaine Parse to str

Un outils de conversion de donnée est ensuite utilisé afin de parser les colonnes **ingredients, étiquette et mode\_emploi** en chaine[DT\_STR] et de taille 800. Leur alias de sortie sont **Copingredients**,**Copetiquette** et **Copemploi** il s’agit des flux de donnée qui serviront par la suite aux traitements.

Une image contenant table

Description générée automatiquement

##### Dérivation des données, Formatage data

Juste après, la ‘Colonne dérivée’ nommée **‘Formatage data**’ permet une première mise en forme des sorties sélectionnées.

* Logo : Affiche si le produit est ‘bio’ ou/et ‘cosmetic’ selon les valeurs des colonnes bio et comestic
* Name : Correspond à la description du produit sans la dénomination ‘ (Vrac) ’
* Category : Affiche les valeurs 2 ou 0, selon s’il s’agit d’une huile végétale ou autre
* Price : Correspond au prix au litre, les mêmes traitements que density sont effectués pour cette colonne.
* Density : Correspond à la densité avec la suppression des espaces superflus et l’ajout d’un 0 lorsque la valeur est inférieure à 1 (qui disparait à l’importation des données)
* pao : si la valeur est null alors elle est remplacée par 0. Ce traitement permet d’éviter des erreurs d’exécutions lorsque la valeur de la colonne pao est null.
* Dlu : à l’origine est de type DataTime puis converti en chaine. De plus, ses valeurs sont au format anglosaxon : Year – Month - Day H : m : s. Le traitement établi ici permet d’extraire la date, le mois ainsi que l’année afin que les valeurs de dlu soient des chaines indiquant la date au format français : Day – Month – Year.
* Copemploi : écriture en brutes des mentions d’emploi si la chaine « yeux » est trouvée. Sinon vide. Correspond aux besoins actuels des étiquettes.

Les Mentions Ecocert ne sont pas actuellement récupérées

* Lot : Si null affiche ‘’’’, sinon affiche le lot.
* Origin : Si null affiche ‘’’’, sinon si « » affiche ‘’’’, sinon affiche origin

L’image ci-dessous montre plus en détails les traitements établis :

Une image contenant texte

Description générée automatiquement

##### Création du flux de données finale : ‘Script lstproducts’

Mise en place

Un script sous Visual Studio 2012 est établi afin de rendre automatique la mise en forme finale des différents fichiers product.txt. Le script récupère en entrée l’ensemble des colonnes disponibles en lecture seule, à savoir :

|  |  |
| --- | --- |
| * no * name, * scrientifc\_description * bio * cosmetic * price * density * logo | * lot * dlu * origin * pao * id * Copingredient. |

En sortie, une seule colonne de sortie est créée**, lstProducts**. Elle récupère l’ensemble des données formatées par le script. Il est nécessaire de préciser son DataType à chaine[DT\_STR]. Enfin, Dans les propriétés communes de la sortie **Sortie0,** il est important de modifier la valeur de la propriétés **SynchronousInputID** afin de la mettre sur **Aucun**. L’image ci-dessus rappelle les différents points précisés.

La sortie doit être encodé en UFT8, ainsi si est nécessaire de vérifier la propriété **CodePage** soit bien à la valeur **65001**

Une image contenant table

Description générée automatiquement

##### Présentation script de formatage

Le script écrit en C# se compose d’un main et d’une classe, respectivement nommées main.cs et Bulk.cs. Le main récupère les différentes valeurs d’un produit vrac via la méthode Entrée0\_ProcessInputRow, les stocke dans une variable dédiée puis appelle le constructeur de la classe Bulk afin de créer une nouvelle entité de bulk. Ensuite, une nouvelle ligne est ajoutée dans la variable de sortie lstBulk où les informations formatées sont stockées via la méthode ToString de la classe Bulk.

|  |
| --- |
| public override void Entrée0\_ProcessInputRow(Entrée0Buffer Row)  // Variable for overload  string no = Row.no;  // Variables for builder  string id = "[" + Row.vracno + "]";  string name = Row.name;  string category = Row.category;  string description = "";  string number = Row.vracno;  string price = Row.price;  string origin = Row.origin;  string density = Row.density;  string lot = Row.lot ;  string logo = Row.logo;  //Variables for description and specifiques mentions  string inci = Row.scientificdescription;  string ingredient = Row.ingredients;  string dlu = Row.dlu;  int pao = Row.pao;  // Variables à modifier  string mention = Row.emploi;  string risks = "";  Bulk newBulk = new Bulk(id, name, category, description, number, price, origin, density, lot, logo, mention, risks);  if (no.Substring(0, 4) == "BASE")  {  newBulk.Description = newBulk.MentionsProducts(ingredient, dlu, mention);  }  if (no.Substring(0, 2) == "HV")  {  mention = "Ne pas avaler. Tenir hors de portée des enfants. Stocker a l abri de la chaleur et de la lumiere.";  if(no == "HV0074V")  {  risks = "Huile puissante, à utiliser diluée. Prudence sur peaux et cuirs chevelus sensibles.";  }    newBulk.Description = newBulk.MentionsProducts(inci, dlu, mention, risks);  }  if (no.Substring(0, 3) == "SAV")  {  newBulk.Description = newBulk.MentionsProducts(ingredient, pao, mention);  }  Sortie0Buffer.AddRow();  Sortie0Buffer.lstProducts = newBulk.ToString();  } |

Le main permet aussi de sélectionner une méthode de gestion de la description selon le type de produit. La propriété description de Bulk est initialisée à vide afin de permettre un traitement de sa description en fonction des mentions obligatoires ou d’utilisation relatif au produit.

De son côté, la classe bulk instancie et valorise les différents produits vrac récupérés ligne à ligne. Les propriétés de la classe Bulk sont les suivantes :

* id, name, category, description, number, price, origin, density lot, logo et mention. Elles sont toutes de type chainée.

De plus deux méthodes sont présentes dans cette classe :

* ToString qui redéfinit la méthode ToString et renvoi une chaine qui contient l’ensemble des données formatées du produit vrac
* MentionProducts retourne une chaine qui contient l’ensemble des descriptions et mentions obligatoire du produit vrac concerné. Cette méthode est surchagé afin de correspondre aux attentes des mentions obligatoires qui divergent selon le type de catégorie de produit, soit :
  + Huile végétale :
    - Une dénomination scientifique de la plante utilisée, **inci** *(string)*
    - Une date limite d’utilisation, **dlu** *(string)*
    - Des mentions de précautions d’emploi, **mention** *(string)*
    - Parfois des risques potentiels, **risks** *(string)*
  + Base :
    - Une liste d’ingrédient, **ingredient.** *(string)*
    - Une date limite d’utilisation, **dlu** *(string)*
    - Des mentions de précautions d’emploi, **mention***(string)*
  + Savon
    - Une liste d’ingrédient, **ingredient** *(string)*
    - Un nombre de mois de péremption après ouverture, **pao** *(int)*
    - Des mentions de précautions d’emploi, **mention** *(string)*

|  |
| --- |
| /// Overload Method  public string MentionsProducts(string mention)  {  return mention;  }  /// <summary>  /// Overload Method for BASE product  /// </summary>  /// <param name="ingredient"></param>  /// <param name="dlu"></param>  /// <param name="mention"></param>  /// <returns></returns>  public string MentionsProducts(string ingredient, string dlu, string mention)  {  string str = "Ingrédients: " + ingredient + "\r\nA utiliser de préf. avant fin : " + dlu + mention;  return str;  }  /// <summary>  /// Overload method for vegetable oil products  /// </summary>  /// <param name="inci"></param>  /// <param name="dlu"></param>  /// <param name="mention"></param>  /// <param name="risque"></param>  /// <returns></returns>  public string MentionsProducts(string inci, string dlu, string mention, string risk)  {  string str = "Ingrédient cosmétique Désignation INCI : " + inci + "\r\nA utiliser de préf. avant fin :" + dlu + " " + mention + risk+ "\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n";  return str;  }  /// <summary>  /// Overload method for soap products  /// </summary>  /// <param name="ingredient"></param>  /// <param name="pao"></param>  /// <param name="mention"></param>  /// <returns></returns>  public string MentionsProducts(string ingredient, int pao, string mention)  {  string str = "Ingrédients : " + ingredient + "\r\nPAO : " + pao + " mois\r\n" + mention;  return str;  }  /// <summary>  /// ToString method for formatted datas  /// </summary>  /// <param name="endrow"></param>  /// <returns>string with data</returns>  public override string ToString()  {  string str = Vracno + "\r\nnom = " + name + "\r\ncategorie = " + category + "\r\ndescription = '''" + description;  str += "'''\r\nlot = " + lot + "\r\nprix = " + price + "\r\nproducteur = " + origin + "\r\ndensite = " + density;  str += "\r\nnumero = " + number + "\r\nlogos = " + logo;  return str;  } |

##### Encoder en UFT-8 Uft8 Encoding

Par l’intermédiaire un nouvel outil de conversion, nommé Uft8 Encoding, le flux de sortie lstProducts va être convertie en chaine unicode [DT\_WSTR]. Son alias de sortie et CoplstProducts

##### Générer les fichiers à envoyer : ‘Destination de fichier plat’

Enfin, une destination de fichier plat est utilisée pour recevoir l’ensemble des données traitées. Dans le gestionnaire de fichier plat, un fichier products.txt contenant une colonne d’entrée **‘lstbulk;’** est utilisé comme fichier de base de destination. Ce dernier est créé lors de l’exécution de processus, **‘Cmd Create product file’** et doit se trouver dans le chemin spécifié par la variable **strFolder**

Dans la fenêtre des propriétés du fichiers plats, dans Expression ; la propriété ‘**ConnectionString’,** doit êtrevalorisé en fonction de l’appareil concerné lors de l’exécution de la boucle **‘device and ip’**. Cette propriété désigne le nom du fichier généré, elle doit être exprimée comme il suit :

**@[User::strFolder]+ @[User::device]+"\_product.txt"**

**Une image contenant texte, capture d’écran, écran

Description générée automatiquement**

Dans l’éditeur de gestionnaire de connexion, le nom de fichier spécifié doit correspondre à celui précédemment créé. **Page de Code** doit être au format **UFT-8 et** il convient ainsi de vérifier que le séparateur correspond à celui utiliser, ici ‘{CR}{LF}’

Une image contenant texte

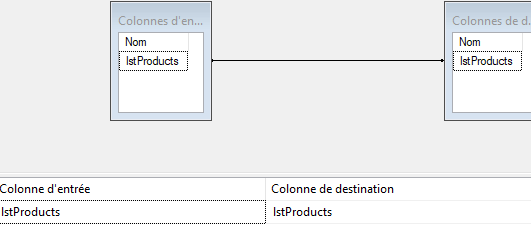
Description générée automatiquement

De plus, il est important de préciser une en-tête qui servira pour l’interface de sélection.

Une image contenant texte

Description générée automatiquement

Puis dans l’onglet mappage, il est important de vérifier que la colonne d’entrée et de destination correspondent soient mappées comme sur l’image suivante.



#### Requête d’envoie Cmd curl request

Le site de JeanBouteille utilise un formulaire avec pour récupérer les fichiers envoyés. La méthode est de type ‘POST’ et l’identifiant est name= ‘myFile’. Dans ce cas, la méthode en C# préconisée pour l’envoi de fichier avec une methode Post est d’utiliser la classe HttpWebClient. Néanmoins, à la suite d’une incompatibilité avec le langage C# sous Visual Studio Microsoft 2012, un autre moyen est utilisé.

Une tache d’execution de processus via l’invite de commande windows. Ainsi, il convient de préciser dans la propriété **Executable** le chemin de l’exécutable, sa valeur est  donc : **C:\Windows\System32\cmd.exe**

La propriété **Argument** représente la ligne de commande à exécuter. Cette propriété contient la valeur de la variable **curl** modifiée dans l’onglet **Expression :**

**"/C curl -v --location --request POST \"http://"+ @[User::ip] +"/liste\_produits\" -F myFile=@\""+ @[User::strFolder]+ @[User::device]+"\_product.txt" + "\""**

De même, la propriété **WorkingDirectory** contient la variable **strFolder**



Une image contenant texte

Description générée automatiquement

### Suppression fichier temporaires

Suite à ce traitement, une tache de système de fichier, ‘Remove tmp folder’ permet la suppression du dossier tmpVrac ainsi que de l’ensemble des fichiers que le package a généré. Dans l’éditeur de tache, la propriété opération est fixée sur ‘Supprimer le répertoire’ et la propriété SourceVariable sur strFolder. On obtient alors, la figure suivante : d

