Documentation Coll Dump Fichier

GYBELS Hugo

29 mai 2015

Table des matières

1 Présentation générale

L'outil Coll Dump Fichier est un outil appartenant à l'entreprise Orange utilisé dans le pôle médiation. Il est utilisé comme aide à l'exploitation de Platine. Platine est une application collectant des comptes-rendus d'appels (CRA), qu'elle va traiter grâce à différents modules (collecte, traitement, distribution), puis de les envoyer vers les Systèmes d'Informations (SI) qui traiteront ces données (facturation, enquêtes juridiques, etc.). Platine prépare donc le traitement des données envoyées par les EEC[1] aux ECC[2] pour faciliter ensuite le traitement des données ; principalement la facturation.

FIGURE 1 – Fonctionnement de Platine

L'application Platine traite des milliards de CRAs tous les jours. Pour gérer un tel flux, tous les CRAs sont convertis dans le format TLV (Type-Length-Value) avant d'être traités par Platine , et seul les champs importants du CRA sont conservés de façon à pouvoir traiter par la suite ces CRAs le plus rapidement possible.

Cependant, certaines fois, on retrouve des CRAs erronés. Lorsque cela arrive, il faut déterminer si l'erreur vient de Platine ou de l'émetteur. Pour cela, il nous faut donc décoder et afficher le message tel que Platine l'a reçu, avant même que le CRA ne soit converti au format TLV. C'est ici qu'intervient l'outil Coll Dump Fichier : il permet de décoder et d'afficher un fichier de collecte pour chaque type de traitement. **Coll Dump Fichier permet ainsi d'afficher le contenu d'un CRA reçu par platine avant qu'il ne soit converti en format TLV ; cela permet d'observer le message exacte ayant été envoyé par l'EEC**. Ceci permet de localiser si le défaut vient en amont de platine ou s'il est du à la collecte ou aux convertisseurs de Platine. Voici ci-dessous un schéma résumant l'emplacement de l'outil Coll Dump Fichier par rapport à l'application Platine.

1. Equipement Emetteur de Comptes-rendus d'appels (émetteur)
2. Equipement Consomateur de Comptes-rendus d'appels (récepteur)

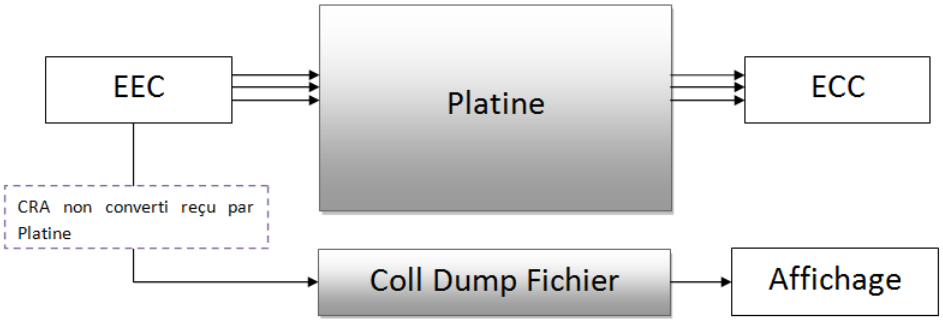


FIGURE 2 – Emplacement de Coll Dump Fichier par rapport à Platine

2 Installation

L'outil Coll Dump Fichier a obligatoirement besoin de trois fichiers pour fonctionner :

2.1 Coll_dump_fichier.ini (Obligatoire)

Le fichier `Coll_dump_fichier.ini` contient les ressources nécessaires au bon fonctionnement de l'outil. C'est-à-dire l'emplacement des fichiers `Coll_dump_traitements.csv`, `instances.csv`, et de l'aide en Français et en anglais. Ci-dessous un exemple du fichier `Coll_dump_fichier.ini` :

```
Liste des traitements :/deci/pood5325/cdf_h/Debug/coll_dump_traitements.csv
Chemin instances :/deci/pood5325/EEC/instances/
Fichier aide Francais :/deci/pood5325/cdf_h/Debug/help_FR/coll_dump_fichier_aide_fr.txt
Fichier aide anglais :/deci/pood5325/cdf_h/Debug/help_EN/coll_dump_fichier_aide_en.txt
```

FIGURE 3 – Coll_dump_fichier.ini

2.2 Coll_dump_traitements.csv (Obligatoire)

Le fichier `Coll_dump_traitements.csv` doit se situer à l'emplacement indiqué dans le fichier Coll_dump_fichier.ini. Il effectue la correspondance entre les différents types de traitements (TTnss9, TTprims, etc.) et la norme qu'ils utilisent (BER, CMM, FIXE, TLV, etc.). Il indique également l'emplacement du fichier contenant le détail des champs du type de traitement (les fichiers de décodage). Il doit être mit à jour pour chaque nouveau type de traitement. Voici un extrait du fichier :

Nom de Traitement	Fichier de decodage	Nombre de lecteurs	
trait1	/deci/pood5325/Outil/coll_dump_traitements/trait1.csv	1	LecteurTLV
TT02F_HP	/deci/pood5325/Outil/coll_dump_traitements/TT02F_HP.csv	1	LecteurFIXE
TTbgw_gsn	/deci/pood5325/Outil/coll_dump_traitements/TTbgw_gsn.csv	1	LecteurBER
TTbgw_gsn	/deci/pood5325/Outil/coll_dump_traitements/TTbgw_gsn.csv	1	LecteurBER
TTbgw_gsnN	/deci/pood5325/Outil/coll_dump_traitements/TTbgw_gsn.csv	1	LecteurBERN
TTcb	/deci/pood5325/Outil/coll_dump_traitements/TTcb.csv	1	LecteurTLV
TTcitR4	/deci/pood5325/Outil/coll_dump_traitements/TTcitR4.csv	1	LecteurTLV
Ttcrae	/deci/pood5325/Outil/coll_dump_traitements/TTcrae.csv	1	LecteurBER
TTgourou	/deci/pood5325/Outil/coll_dump_traitements/TTgourou.csv	1	LecteurTLV
TTincc	/deci/pood5325/Outil/coll_dump_traitements/TTincc.csv	1	LecteurCMM
TTmib	/deci/pood5325/Outil/coll_dump_traitements/TTmib.csv	1	LecteurCSV
TTmmm	/deci/pood5325/Outil/coll_dump_traitements/TTmmm.csv	1	LecteurTLV

FIGURE 4 – Coll_dump_traitements.csv

2.3 Fichiers de décodage : traitement.csv (Obligatoires)

Chaque type de traitement doit correspondre à un fichier de décodage de la forme `Traitement.csv`. Ce fichier doit se situer à l'emplacement indiqué dans la ligne correspondante du fichier Coll_dump _traitements.csv. Ce fichier contient l'intégralité des champs du DC ainsi que le numéro de la fonction de décodage du champ (exemple : date, ACSII, numéro de téléphone etc.) et le décalage de l'affichage. Pour finir il contient la forme du champ : si le champs est un primitif (une valeur à afficher) le champ contient la valeur "NULL", si c'est un constructed (un autre sous tableau) il contient le nom du sous tableau suivant.

Voici un tableau qui résume la structure d'un fichier csv :

Tag	Nom du champ	Décalage de l'affichage	Numéro de la fonction de décodage	Nature du champ suivant	0

Les éléments de chaque ligne du fichier csv doivent être séparés par des ";". Ci-dessous un exemple de fichier csv (pour le TTnss9) avec quelques sous tableaux.

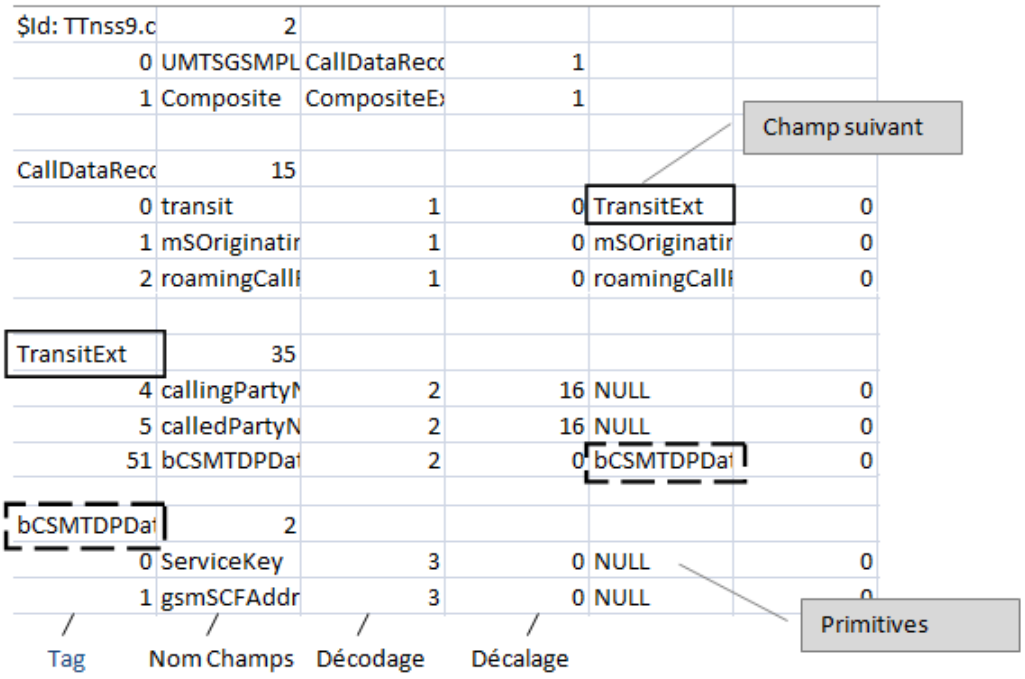


FIGURE 5 – Exemple de fichier csv (TTnss9.csv)

2.4 Dossier Instances (Facultatif)

Le dossier instances contient les descriptifs de toutes les EEC. Il permet de retrouver le type de traitement d'une EEC sans qu'on ait à le préciser dans la commande. Pour cela il faut juste qu'on retrouve le numéro de l'EEC dans le nom du fichier de collecte. Il doit se trouver à l'emplacement indiqué dans Coll_dump_fichier.ini.

Ce dossier est donc **obligatoire sans l'option -t** *cf. Partie 3.2 Commandes.*

```
[global]
AX_IDENT_EEC    = "WIFI"
AX_EEC_PROFIL   = "PUSH_CFT"
AX_EEC_PROTO    = "PUSH-CFT"
AX_EEC_CDATE    = 1257172699
AX_EEC_VALID    = 1
AX_LIBELLECOMPL_EEC     = "Fusion Work WIFI"
AX_EEC_FREQTRANSFERT    = 100
```

FIGURE 6 – fichier instances\1.ini : description de l'EEC 1

L'EEC 1 envoie donc des CRAs sous le type de traitement WIFI.

2.5 Les fichiers d'aide (Facultatifs)

Les fichiers d'aides en Français comme en Anglais sont des fichiers text (.txt). Il doivent avoir le nom et l'emplacement indiqué dans le fichier Coll_dump_fichier.ini. Leur rôle est de résumer l'utilisation de l'outil ainsi que des différentes options (détail plus tard dans la doc).

3 Utilisation

3.1 Environnement de Travail

Gestion de configuration

Le code source de Coll dump fichier est géré grâce à une gestion de versions sous éclipse. On utilise pour cela un dépôt sous Orange forge et un client SVN sous éclipse. Avant tout développement majeur, il est conseillé de créer une branche séparée du tronc que l'on mergera plus tard si les modifications sont à conserver. Pour cela on utilise un autre client directement intégré à Windows : *Tortoise* plus performant et plus convivial que le client éclipse. Par exemple, voici la branche SVN que je me suis créé pour mes travaux :

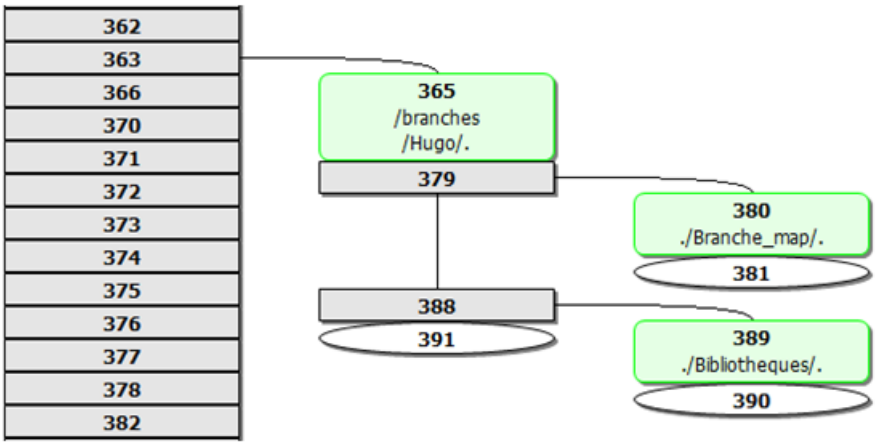


FIGURE 7 – Graphique de révision svn de CDF.

Il est conseillé de créer une copie de travail locale lorsque l'on effectue un checkout du projet.

Création d'un projet éclipse

Il faut ensuite créer un projet SVN sous éclipse. Pour cela, il faut créer un nouveau projet sur la machine distante grâce au menu *File ->New ->Other...->SVN ->Checkout Projetcs from SVN* (le client SVN doit être installé). Le lien UML du dépôt est le suivant :

https ://www.forge.orange-labs.fr/svnroot/coll-dump-fichier

On peut ensuite récupérer les fichiers sources. Il est ensuite nécessaire de compiler une fois sous éclipse de façon à générer les fichiers Makefile. ATTENTION : on va exécuter le programme sous linux, on utilise donc pas le compilateur par défaut mais bien **Cross GCC** disponible dans le compilateur MinGW32.

Exécution sur la machine distante

Une fois les Makefiles générés, on peut compiler sous la machine distante. Pour cela on ouvre un putty. Ci-dessous une capture d'écran de la configuration de la machine distante (DEV22).

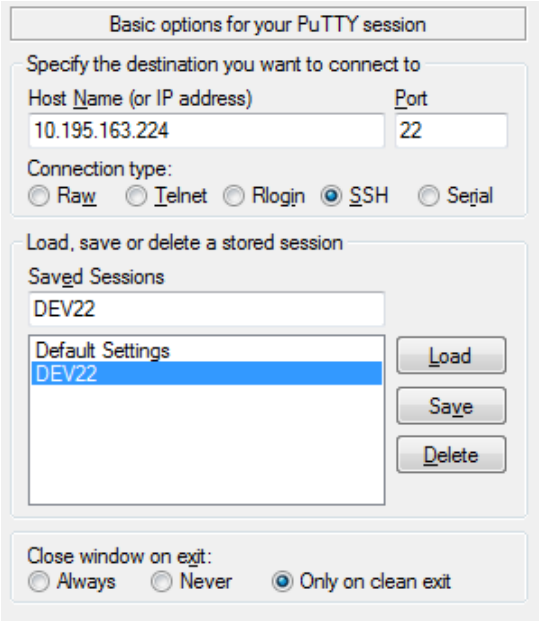


FIGURE 8 – Configuration de la machine DEV22.

Une fois le terminal ouvert, il ne sera plus nécessaire de compiler sous éclipse mis à part lors de la création de nouveaux fichiers. Pour compiler, on se place donc dans le répertoire Debug créer dans le dossier contenant notre projet. On exécute ensuite la commande `make clean` puis la commande `make` qui lancera la compilation du projet.

Le lancement de l'exécutable sera détaillé par la suite.

3.2 Commandes

Une fois installé, nous pouvons décoder différents types de fichiers de collecte. Dans tous les cas, la syntaxe est la même pour la ligne de commande :

coll_dump_fichier [-f] 'emplacement_du_fichier_de_collecte' [-options]

Il existe ensuite plusieurs options permettant de décoder différemment le fichier de collect. Voici ci-dessous un tableau résumant ces options.

-h + -fr/-en	Affiche l'aide
-f	Obligatoire pour toutes les options ci-dessous.
-t	Force le type de traitement. Doit être suivi du type de traitement.
-v	Affiche les DC seulement.
-d	Affiche le détail du DC (ne peut être utilisé qu'avec l'option -v).
-i	Permet d'ajouter l'identifiant d'audit (ne peut être utilisé qu'avec l'option -v).
-e	Extraction du fichier spécifié.
-s	Analyse un fichier sans entête platine (le fichier contient seulement un DC). On doit préciser le type de traitement avec -t.
-stat	Affiche le nombre de CRA dans le fichier.

FIGURE 9 – Liste des options de commande.

Voici deux exemples de ligne de commande. Dans la première, le type de traitement n'est pas précisé, il le trouve donc grâce au numéro d'ECC ; et dans la 2ième c'est un fichier sans entête qui n'est pas forcément un fichier de collecte dont on précise le type de traitement manuellement.

```
coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/coll_0005_120214_09_06_001 -v -d
```

```
coll_dump_fichier -f /deci/pood5325/collecte/fichiersCollecte/CACIVCMFRAF103275 -t TTcrae -v -d -s
```

FIGURE 10 – Exemples de ligne de commande.

Voici le résultat que l'on obtient sur ce type de commande :

FIGURE 11 – Exemples de résultat de Coll Dump Fichier.

4 Fonctionnement du code

Cette partie à pour but d'expliquer le fonctionnement global du code : quelles sont les grandes étapes, comment les données sont stockées, décodées et affichées. Encore une fois, on gardera beaucoup de recul sur le fonctionnement du code, on n'explicitera pas les détails du fonctionnement.

4.1 Fonctionnement général

Nous avons déjà vu dans quel contexte l'outil Coll Dump Fichier est utilisé, mais avant de parler du code lui même, voici un résumé du fonctionnement de ce dernier.

Pour commencer, un rappel sur le principe du code : il doit ouvrir un fichier de collecte (qui contient des comptes-rendus d'appels, ou CRAs). Il va ensuite récupérer le type de traitement de l'EEC, autrement dit, le format ou la norme dans laquelle est codé le compte-rendu. Suite à cela, il va isoler chaque compte-rendu, qu'il va ensuite afficher dans la console avec le détail des champs qu'il contient. Cette étape est justement automatique dans Platine, qui elle transmet automatiquement les DC sans même les afficher.

Ces comptes-rendus sont donc composés d'une listes de champs correspondants à des valeurs. Par exemple, on peut retrouver le champ "Durée de l'appel" avec la valeur 60 secondes. Cependant, dans le fichier on aura pas le nom du champ "durée de l'appel", mais un numéro (ou tag) par exemple le tag 10 qui identifie ce champ. Il faut donc aller chercher la signification du numéro du champ dans le fichier csv qui contient la correspondance entre les tag et le nom des champs. Toujours dans notre exemple, l'outil va chercher, dans le fichier csv du type de traitement, la case avec le tag 10 qui correspondra au nom de champ "durée de l'appel" et le type de données sera "secondes". Voici un schéma qui résume ce fonctionnement :

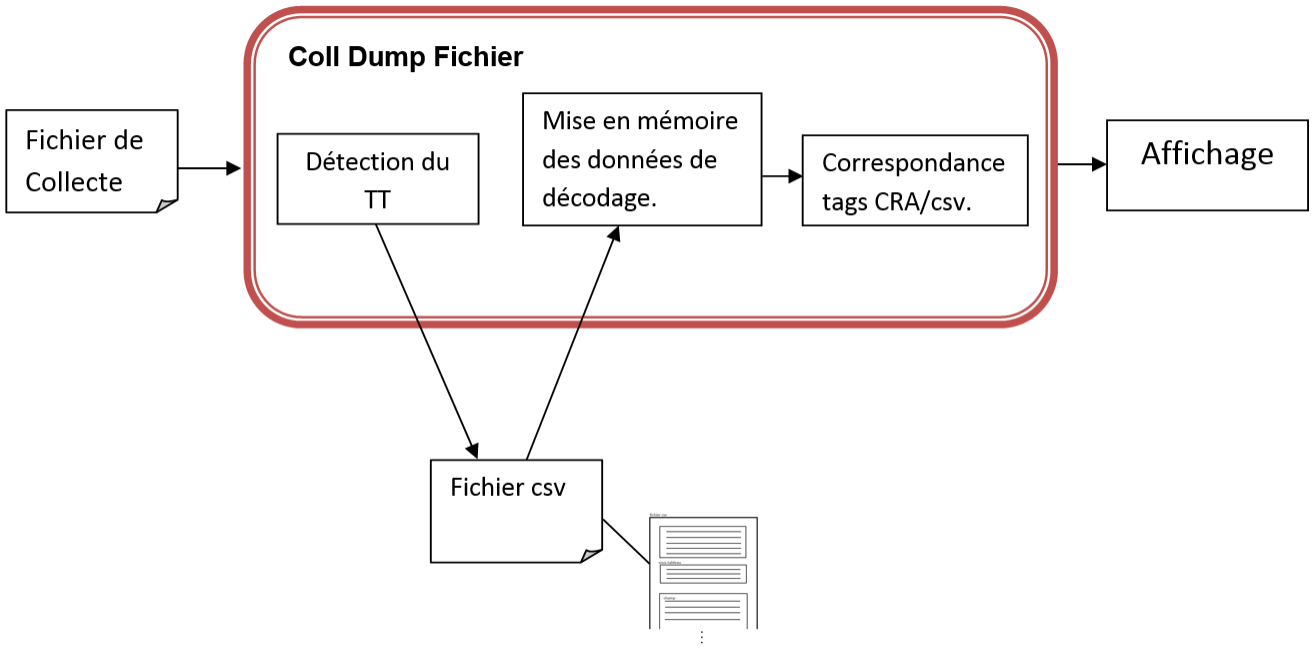


FIGURE 12 – Fonctionnement du code de Coll Dump Fichier.

Pour rendre cela plus concret, voici un exemple d'utilisation quotidienne de l'outil. La ligne de commande peut être combinée avec un ou plusieurs grep, ou encore sa sortie peut être redirigée vers un autre fichier (*Cf. figure 13*).

4.2 Lecture du fichier csv

4.2.1 Instanciation de la classe Lecteur

L'instanciation de la classe Lecteur se fait grâce aux classes *RecTraitement* et *Traitement* appelées dans le main(). En effet, la classe *RecTraitement* récupère le nom du ou des lecteur(s) dont nous allons avoir besoin pour le décodage. Il a deux façon de récupérer ce type de traitement : soit il est indiqué dans la ligne de commande à la suite de l'option `-t` ; soit le nom du fichier de collecte contient le numéro de l'EEC et il trouvera le type de traitement correspondant grâce au fichier `instances.csv`. La déclaration des lecteurs se fait dans une instance de type traitement à laquelle on ajoute tous les lecteurs dont nous avons besoin.

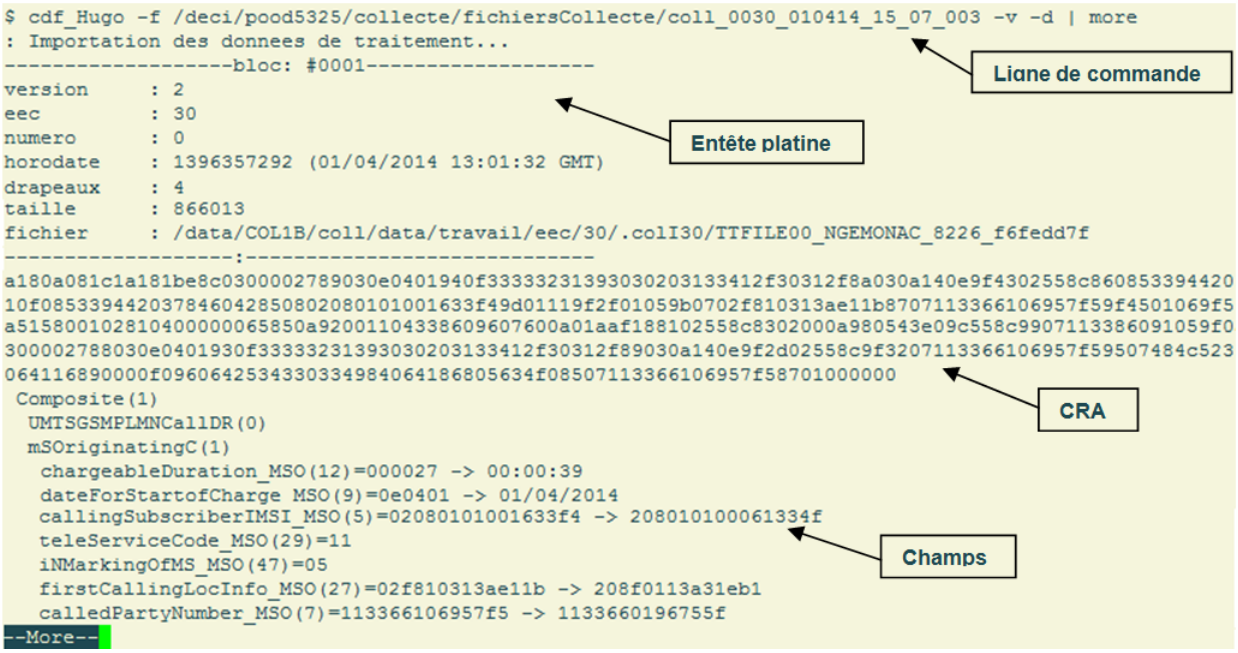


FIGURE 13 – Exemple d'utilisation classique de Coll Dump Fichier.

4.2.2 Structures de stockage des données

Le stockage des données contenues dans le fichier *.csv* se fait dans le constructeur de la classe fille du Lecteur appelé (par exemple dans LecteurBER). On prendra par la suite l'exemple du LecteurBER afin de mieux comprendre, mais la structure est la même pour les autres lecteurs. Les données du csv sont stockées dans plusieurs tableaux dynamiques (vector) appartenants à plusieurs classes. Le graphique ci-dessous résume l'appel de ces classes par ordre chronologique :

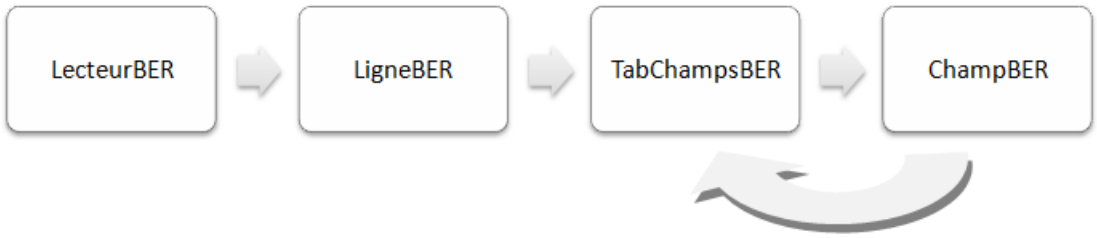


FIGURE 14 – Appel des classes de stockage du csv.

Chacune de ces classes contient un tableau dynamique correspondant à une partie du fichier csv, le tableau ci-dessous résume ce à quoi correspond chacune des classes :

LecteurBER	LecteurBER contiendra donc l'integralité du fichier csv dans *p_ligne*.	*p_ligne* : Tableau de LigneBER.
LigneBER	Chaque LigneBER correspond à une "nature" du DC. Il y aura autant de LigneBER que de ligne de 4 colonnes au début du fichier csv.	*p_tab_champs* : Tableau de TabChampsBER.
TabChampsBER	Chaque champ de forme *constructed* correspond à un TabChampsBER. TabChampsBER correspond donc à un sous tableau du fichier csv.	*p_champs* : Tableau de ChampBER
ChampBER	ChampBER contient les données de chacune des cases du fichier csv. ChampBER correspond donc à une ligne du csv.	*p_tag*, *p_flag*, *p_decodage*, etc.

Voici un exemple de cette structure sur un fichier csv correspondant à un type de traitement (ici c'est le fichier csv du TTnss9) :

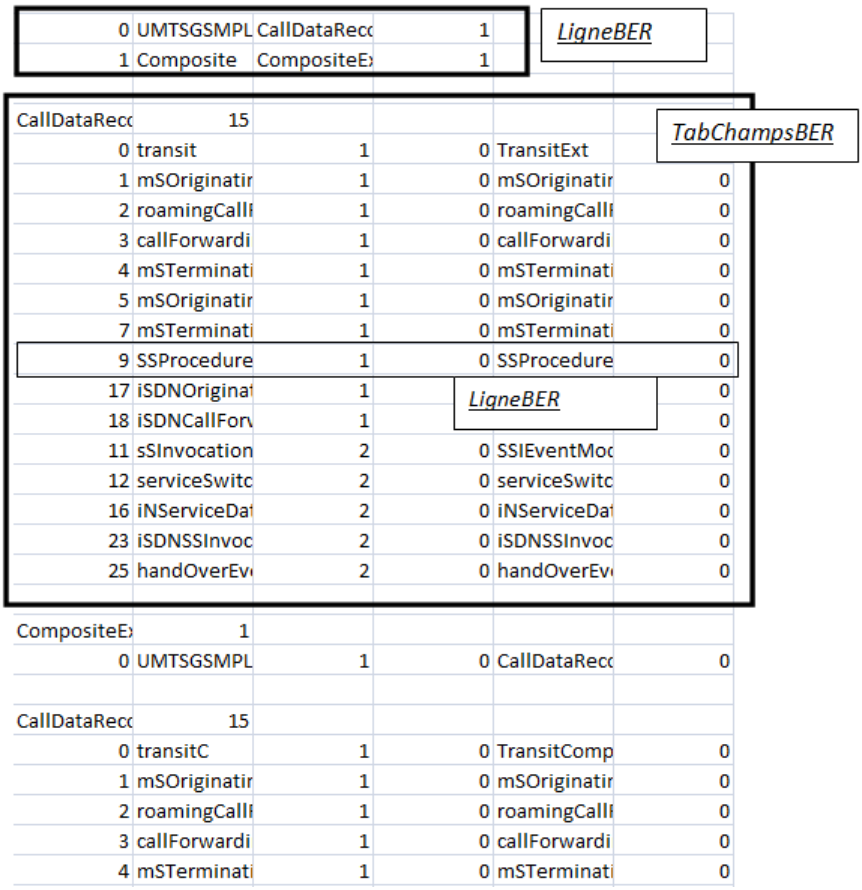


FIGURE 15 – Structuration d'un fichier csv.

4.3 Decodage du fichier de collecte

Une fois toutes les données nécessaires au décodage des DC chargées en mémoire ainsi que le fichier de collecte, on peut commencer le décodage ainsi que l'affichage des différents DC. L'intégralité du décodage se déroule dans le fichier Decodeur (par exemple DecodeurBER). La fonction **Decode** analyse une ligne et appel la fonction **AnalyseChamp** qui va soit afficher la valeur du champs si c'est une primitive, soit s'appeler à nouveau jusqu'à tomber sur un champs primitif. C'est une fonction récursive qui va nous permettre de parcourir l'intégralité du fichier csv stocké en mémoire en effectuant des comparaison sur le tag des lignes du DC et du fichier csv. Pour finir, la fonction AnalyseChamp fait appel à une dernière fonction : la fonction **DecodeChamp** pour mettre en forme la valeur du champ avant de l'afficher ; par exemple, si c'est une date il la met au format aaaa.mm.jj, de même pour les IPv6, etc.

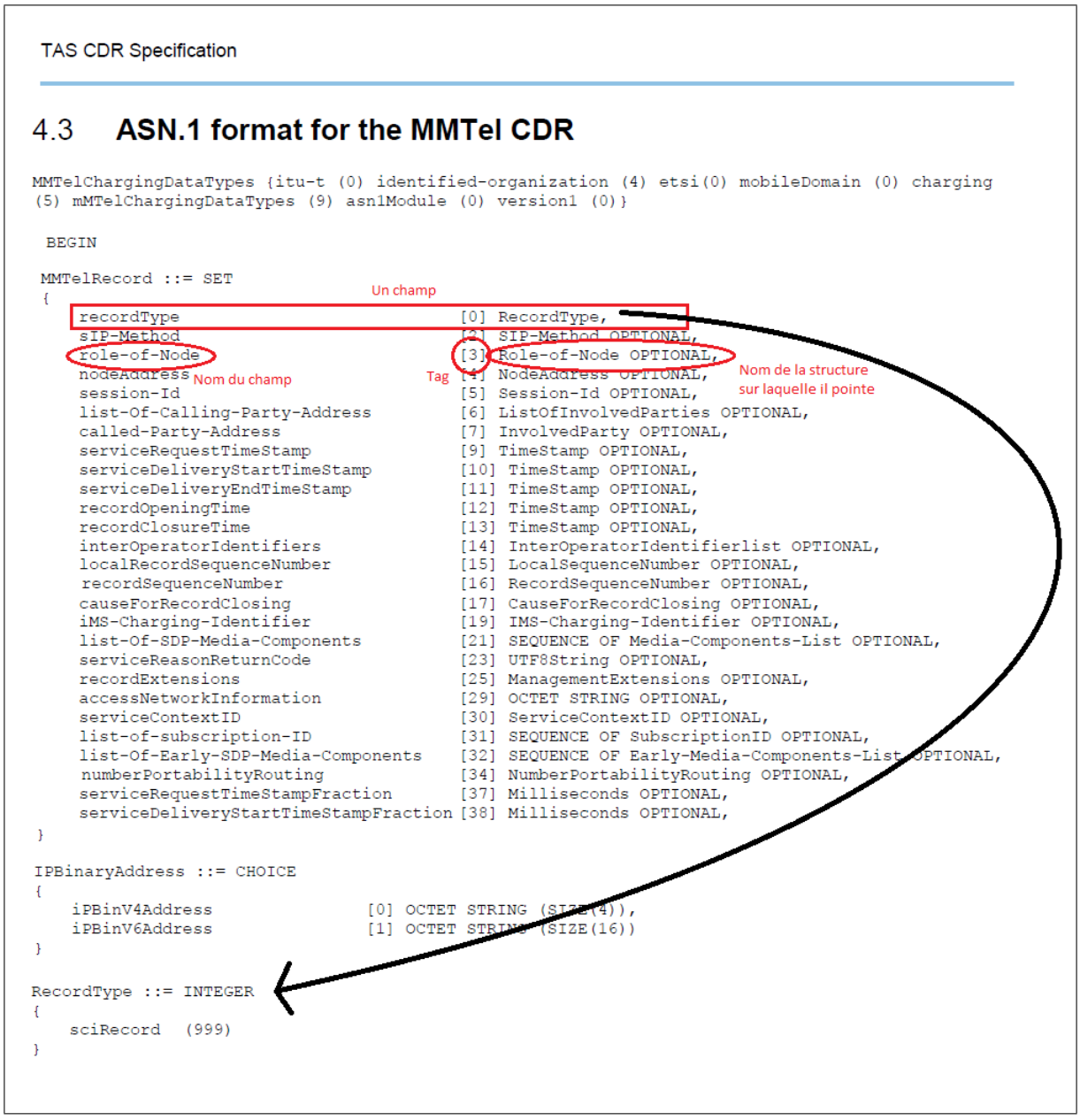
Remarque : On ne passe par cette partie seulement si on à choisi l'option -d correspondant au décodage.

5 Maintenance

5.1 Construction du fichier csv

Lorsqu'un nouveau type de traitement apparait, il est nécessaire de créer le fichier csv correspondant au type de traitement. Pour le réaliser on utilise la **documentation constructeur**. On va l'utiliser pour retrouver tous les champs qui peuvent être reçus.

Il faut donc rechercher dans la documentation constructeur la liste des champs ainsi que leurs tags, et le type de données qu'il contiennent. Prenons l'exemple du TTtas.csv. Voici ci-dessous un extrait de la partie lisant les champs de la documentation TTtas :




```
MMTelChargingDataTypes {itu-t (0) identified-organization (4) etsi(0) mobileDomain (0) charging
(5) mMTelChargingDataTypes (9) asn1Module (0) version1 (0)}

 BEGIN

MMTelRecord ::= SET
{
    recordType                          [0] RecordType,
    sIP-Method                          [2] SIP-Method OPTIONAL,
    role-of-Node                        [3] Role-of-Node OPTIONAL,
    nodeAddress                         [4] NodeAddress OPTIONAL,
    session-Id                          [5] Session-Id OPTIONAL,
    list-Of-Calling-Party-Address       [6] ListOfInvolvedParties OPTIONAL,
    called-Party-Address                [7] InvolvedParty OPTIONAL,
    serviceRequestTimeStamp             [9] TimeStamp OPTIONAL,
    serviceDeliveryStartTimeStamp       [10] TimeStamp OPTIONAL,
    serviceDeliveryEndTimeStamp         [11] TimeStamp OPTIONAL,
    recordOpeningTime                   [12] TimeStamp OPTIONAL,
    recordClosureTime                   [13] TimeStamp OPTIONAL,
    interOperatorIdentifiers            [14] InterOperatorIdentifierlist OPTIONAL,
    localRecordSequenceNumber           [15] LocalSequenceNumber OPTIONAL,
    recordSequenceNumber                [16] RecordSequenceNumber OPTIONAL,
    causeForRecordClosing               [17] CauseForRecordClosing OPTIONAL,
    iMS-Charging-Identifier             [19] IMS-Charging-Identifier OPTIONAL,
    list-Of-SDP-Media-Components        [21] SEQUENCE OF Media-Components-List OPTIONAL,
    serviceReasonReturnCode             [23] UTF8String OPTIONAL,
    recordExtensions                    [25] ManagementExtensions OPTIONAL,
    accessNetworkInformation            [29] OCTET STRING OPTIONAL,
    serviceContextID                    [30] ServiceContextID OPTIONAL,
    list-of-subscription-ID             [31] SEQUENCE OF SubscriptionID OPTIONAL,
    list-Of-Early-SDP-Media-Components  [32] SEQUENCE OF Early-Media-Components-List OPTIONAL,
    numberPortabilityRouting            [34] NumberPortabilityRouting OPTIONAL,
    serviceRequestTimeStampFraction     [37] Milliseconds OPTIONAL,
    serviceDeliveryStartTimeStampFraction [38] Milliseconds OPTIONAL,
}

IPBinaryAddress ::= CHOICE
{
    iPBinV4Address              [0] OCTET STRING (SIZE(4)),
    iPBinV6Address              [1] OCTET STRING (SIZE(16))
}

RecordType ::= INTEGER
{
    sciRecord   (999)
}
```

FIGURE 16 – Structuration d'un fichier csv.

Il faut ensuite remplir le fichier *traitement.csv* selon le tableau décrit précédemment (*Cf. Fichiers de décodage : traitement.csv*). Lorsque le champ contient une valeur (un nombre ou une date) on remplie le champs *Nature du champ suite* par "Null" et s'il est du type constructed (s'il pointe sur un autre tableau) on indique simplement le nom du tableau sur lequel on pointe.

Remarque : On peut ajouter des commentaire dans un fichier csv en début la ligne à commenter par un point virgule (';').