PMSIXML

Table des matières

Pmsixml est un projet dont le but est de rendre l'analyse et le traitement des fichiers PMSI aussi facile que de traiter un fichier XML.

Introduction

Le PMSI a des origines très lointaines (début des années 80), et à l'époque la façon la plus efficace de traiter les fichiers était d'avoir une structure à *enregistrements fixes*. Ce format d'enregistrements à largeur fixe est très efficace, et très rapide à traiter. Mais il manque cruellement de flexibilité ; si l'on décale un enregistrement, *cela décale tous les enregistrements suivants*, et il faut réécrire les programmes pour s'adapter au nouveau format. Pour les traitements statistiques ponctuels, il est suffisant et on entre manuellement les nouvelles positions lorsque les formats changent. De plus la plupart des programmes d'analyse statistiques ont été prévus pour pouvoir traiter ce type de données, on s'en est donc accomodé jusqu'à maintenant.

Mais si l'on veut *automatiser* les traitements, soit on utilise R et il y a des packages pour charger les métadonnées, soit on part de zéro (c'est à dire faire du copier-coller depuis les fichiers excel de l'ATIH). C'est ce qui a été fait avec PmsiXml (excepté qu'en 2016 l'ATIH ne publiait que des fichiers PDF), et en le mettant en publication open source, nous espérons que cela évitera à d'autres d'avoir à repartir de zéro.

La structure d'un enregistrement PMSI

Un enregistrement PMSI est une suite de caractères qui tient sur une ligne. Un champ de cet enregistrement est un nombre fixe de caractères, qui est à un endroit fixe.

La description des champs et des endroits où ils peuvent être trouvés est maintenant publiée par l'ATIH annuellement, par exemple pour l'année 2024 on retrouve les formats à cette adresse : https://www.atih.sante.fr/formats-pmsi-2024-0 . Il y 10 ans la situation était plus compliquée, pour

avoir ces formats il fallait parcourir des documents disparates et publiés au format PDF, et recopier ces descriptions de champs à la main. Mais même encore maintenant, ces descriptions ne sont pas disponibles de façon à être analysées directement par un programme.

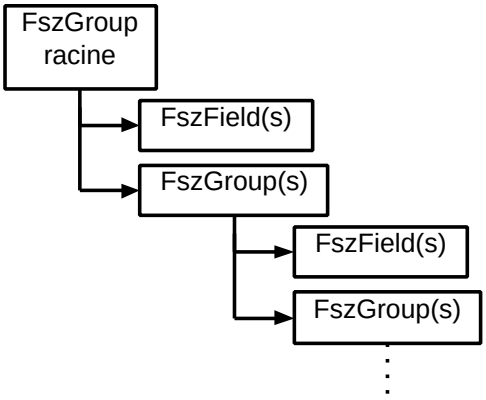
Voici par exemple un extrait du fichier `formats_mco_2024.xlsx`, dans l'onglet "RSS non groupé format 022" :

Libellé		Taille	Début	Fin	Type de données	Précision (type de données)	Caractère obligatoire	Cadrage/ Remplissage
Numéro FINESS d'inscription ePMSI		9	1	9	A	Référentiel FINESS e-PMSI (Plage)	O	NA/NA
Version du format du RUM		3	10	12	A	Valeur fixe	O	NA/NA
N° de RSS		20	13	32	A*		O	Gauche/Espace
N° Administratif local de séjour	(Equivalent de HOSP-PMSI)	20	33	52	A*		O	Gauche/Espace
N° de RUM		10	53	62	A*		O	Gauche/Espace
Date de naissance		8	63	70	Date	JJMMAAAA	O	NA/NA
Sexe		1	71	71	A	Liste	O	NA/NA

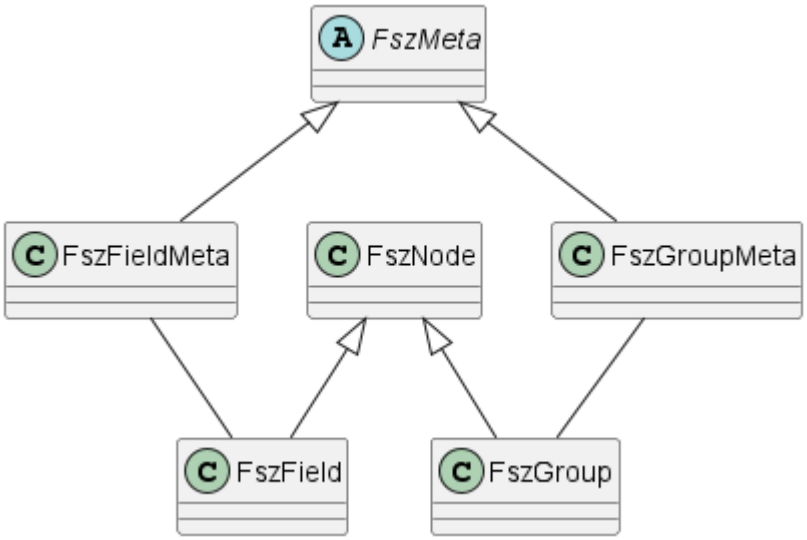
On voit bien qu'il y a des champs fusionnés, des cellules à plusieurs lignes, en gros tout cela n'obéit pas à une structure prévisible et facile à exploiter informatiquement.

Modélisation des éléments PMSI

Le modèle objet des éléments PMSI est juste suffisant pour modéliser les éléments textuels courants que l'on rencontre dans les enregistrements à position fixe du PMSI. L'abréviation Fsz désigne "Fixed-Size" pour dire "position fixe". Les données du PMSI sont rangées dans des champs (FszField) et des groupes de champs (FszGroup), qui forment une arborescence :



Les définitions de champs et de groupes sont lues dans des objets Meta. Ces objets Meta sont ensuite utilisés lors de la lecture des fichiers de données. Voici un bout de la hiérarchie :



Les métadonnées sont lues par un objet *MetaFileLoader*.

L'objet MetaFileLoader est configuré avec un chemin de répertoire dans lequel il va chercher les fichiers de métadonnées. Si le fichier de métadonnées n'est pas trouvé, il va chercher dans les fichiers *resource* qui sont dans le *jar* de pmsixml. Ainsi une définition locale a toujours priorité sur une définition de pmsixml, vous pouvez donc faire vos propres métadonnées si cela est nécessaire (changement du nom compact, ajout de champs, etc.)

La lecture des données est faite par un objet *FszReader*. Cet objet va utiliser une *stratégie* pour lire les métadonnées, puis les données. La stratégie va déterminer notamment comment lire les métadonnées ; par exemple pour un RSS, il faut lire la version dans les caractères 9 à 12, et ensuite on saura quel fichier de métadonnées il faut charger. Avec le bon fichier de métadonnées chargé, la lecture s'effectuera correctement normalement. L'objet stratégie donne aussi la stratégie de lecture ; par exemple pour le RSS on commence par lire les champs ; ensuite on lit les sous-groupes "DA" (en utilisant le compteur "DA" pour savoir combien en lire), puis les sous-groupes "DAD" (compteur "DAD"), puis les sous-groupes "ZA" (compteur "ZA")

La structure des métadonnées PMSI dans PmsiXml

Dans Pmsixml, les métadonnées PMSI sont enregistrées au format .ods (format table Libre Office), et au format .csv, avec un un nombre fixe de colonnes. Le but est d'avoir des métadonnées que l'on peut placer directement à côté du fichier PDF ou du fichier excel distribué par l'ATIH, et de pouvoir ainsi faire la vérifiation rapidement et fiablement, voire même de faire du copier-coller.

Voici un extrait du contenu du fichier `rss022.ods` qui a été construit à partir des métadonnées publiées par l'ATIH :

A	B	C	D	E	F	G	H	I	J	K	L	M
Typ	Libellé	Nomc	Taille	Début	Fin	Obligatoire[1]	Type[2]	TypePre	Cadrage/Remplissage[3]	Remarques	Compteur	Format
RUM	Numéro FINESS d'inscription ePMSI	FINESS	9	1	9	O	A	A	NA/NA			
RUM	Version du format du RUM	VRUM	3	10	12	O	A	A	NA/NA	022		
RUM	N° de RSS (Equivalent de HOSP-PMSI)	NRSS	20	13	32	O	A	A	Gauche/Espace			
RUM	N° Administratif local de séjour (Equivalent de HOSP-PMSI)	NADL	20	33	52	O	A	A	Gauche/Espace			
RUM	N° de RUM	NRUM	10	53	62	O	A	A	Gauche/Espace			

Quelques colonnes ont été ajoutées par rapport à ce qu'on trouve à l'ATIH :

Colonnes supplémentaires

- **Typ** : un code de type d'enregistrement ou de sous-enregistrement
- **Nomc** : "nom court" : donne un nom unique qui permet de désigner ce champ. (Ce nom est arbitrairement attribué, il n'a rien à voir avec l'ATIH)
- **TypePref** : le type préféré pour le champ. Permet de mettre un type différent que celui donnée par l'ATIH. Introduit aussi un type qui n'est pas donnée par l'ATIH, le type D (pour les dates)
- **Remarques** : les remarques qui sont sur le document ATIH
- **Compteur** : lorsqu'il y a un compteur de champs,
- **Format** : donne des informations supplémentaires sur le format du champ.

Voici l'ensemble des colonnes qui doivent être présentes dans un fichier de métadonnées pour un format PMSI :

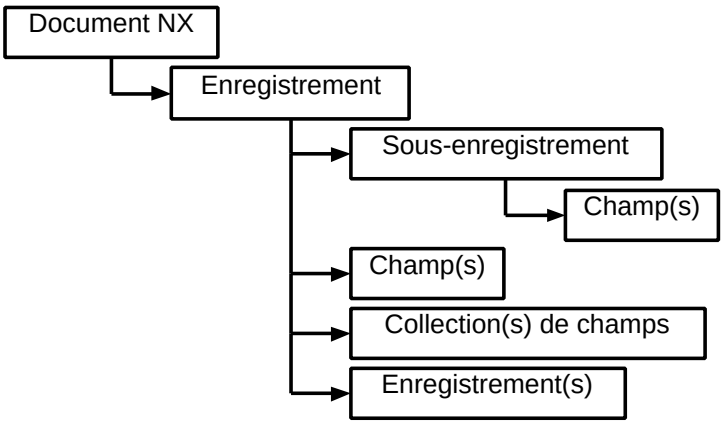
1. Typ : Nom du groupe de champs (voir javadoc de `fr.gpmsi.pmsixml.FszNode`)
2. Libellé : Nom long du champ tel qu'on le trouve dans la doc ATIH
3. Nomc : Nom compact du champ, idéalement moins de 10 caractères, juste des lettres et chiffres, sera utilisé comme nom de variable et de colonne
4. Taille : nombre de caractères du champ, repris de l'ATIH
5. Début : numéro de la colonne du 1er caractère du champ (commence à 1), repris de l'ATIH
6. Fin : numéro de la colonne du dernier caractère du champ (commence à 1), repris de l'ATIH
7. Obligatoire[1] : O ou N, repris de l'ATIH
8. Type[2] : N pour nombre, A pour alpha, repris de l'ATIH
9. TypePref : N pour nombre, A pour alpha, D pour date JJMMAAAA
10. Cadrage/Remplissage[3] :
11. Remarques : remarques données par l'ATIH. Doit tenir sur une ligne. Si on doit indiquer un passage à la ligne, mettre "\n", ce sera remplaçé par un passage à la ligne dans la plupart des outils qui utilisent les métadonnées
12. Compteur : nom qui est utilisé par l'analyseur pour stocker le nombre qui se trouve dans ce champ. Ce compteur sera ensuite rappelé pour avoir le nombre d'éléments à lire.
13. Format : vide ou x+y pour indiquer position d'un nombre à virgule (par ex. 5+2) correspond à l'info donnée par l'ATIH ou la norme B2

Les métadonnées pour le format NX

Le format NX (produit par AMELI, l'assurance-maladie obligatoire) est également un format où les champs occupent une position fixe, mais sa complexité est bien plus grande que les formats PMSI.

Les métadonnées dans Pmsixml pour le format NX sont entrées dans un format XML, qui décrit chaque champ dans chaque enregistrement, mais précise également comment chaque enregistrement doit être ajouté, par rapport aux enregistrements précédents.

Voici en gros l'organisation des éléments modélisés :



C'est encore un travail en cours ; les définitions risquent encore de changer.

Chargement des métadonnées

Les classes de Pmsixml chargent les métadonnées lorsqu'elles en ont besoin.

Pour cela on donne à la classe un répertoire dans lequel rechercher la métadonnée nécessaire. Si cette métadonnée n'est pas retrouvée dans le répertoire, on la cherche alors en "*resource*", c'est à dire dans les fichiers qui ont été distribués avec pmsixml.

Ces fichiers de métadonnées sont dans le fichier jar de pmsixml (par ex. pmsixml-3.1.0.jar) , dans le sous-répertoire `fr\gpmsi\pmsixml\`

Dans la majorité des cas, il n'est pas nécessaire de fournir votre fichier métadonnées, les fichiers qui sont en *resource* sont suffisants. Mais la possibilité existe, et permet par exemple de donner un format qui n'aurait pas encore été fourni par Pmsixml (il en manque encore pas mal à vrai dire, si il y a des volontaires pour s'occuper des fichiers HAD et PSY par exemple ...).

Exemples d'utilisation de PmsiXml

Application démo

Dans les sources il y a une petite classe de démo d'utilisation de lecture de Rss pour montrer une utilisation simple de lecture de RSS :

```
package fr.gpmsi.pmsixml;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * Démonstration très simple d'utilisation de RssReader.
 * Lit un fichier de RUMs/RSS et imprime pour chaque ligne (RUM) le numéro de RUM, le
```

```
numéro de RSS, le numéro de dossier administratif.
 *
 */
public class RssReaderDemo {

    /**
     * Mini application de démo pour analyser un fichier de RSS (groupés ou non)
     * @param args Il ne doit y avoir qu'un seul argument, le chemin du fichier à
analyser
     * @throws IOException si erreur E/S
     * @throws FieldParseException Si erreur dans les métadonnées
     * @throws MissingMetafileException Si pas de métadonnées trouvées pour un RUM/RSS
     */
    public static void main(String[] args)
    throws IOException, FieldParseException, MissingMetafileException
    {
        RssReader rdr = new RssReader();
        String fichierRss = args[0];
        try (FileReader fr = new FileReader(fichierRss)) {
            BufferedReader br = new BufferedReader(fr);
            System.out.println("Num.dossier;Num.RSS;Num.RUM");
            String rss;
            int lineNr = 1;
            while ((rss = br.readLine()) != null) {
                FszGroup gn = (FszGroup) rdr.readOne(rss, lineNr);
                String nrss = gn.getChildField("NRSS").getValue();
                String nrum = gn.getChildField("NRUM").getValue();
                String nadl = gn.getChildField("NADL").getValue();
                System.out.println(nadl+";"+nrss+";"+nrum);
                lineNr++;
            }
        }
    }

}
```

Voici en gros ce que fait l'application :

Elle crée un objet `RssReader`

Pour chaque ligne du fichier Rum/Rss, elle appelle le lecteur de RSS, en lui passant la ligne qui vient d'être lue. Le lecteur de Rss, si tout s'est bien passé, ramène un objet `FszGroup` qui va contenir tout ce qui a été lu.

On peut interroger cet objet pour récupérer les infos de chaque champ, en utilisant son nom compact.

Par exemple si on veut le numér de RSS on va rechercher le champ appelé "NRSS", en faisant `gn.getChildField("NRSS")`. Cet appel envoie un objet qui représente le champ; pour accéder à la valeur brute, il faut encore appeler `getValue()` , d'où la séquence complète d'appel qui est : `gn.getChildField("NRSS").getValue()`

Les appels sont les mêmes pour le numéro de RUM ("NRUM") et le numéro de dossier ("NADL" car dans les docs ATIH et Ameli c'est "numéro administratif de dossier local", mais dans les hôpitaux on utilise aussi IEP, NDA, NDOSS, etc. Je suis resté sur NADL).

Tests de la librairie

Dans le répertoire source src\test\java il y a les tests utilisés pour un certain nombre d'objets de la librairie, ils montrent les utilisations possibles de la librairie.

Javadoc

La référence de documentation est toujours dans la documentation javadoc des classes de la librairie.