

Transformer du XML en CSV

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 Décembre 2006. Dernière mise à jour le 23 Janvier 2007

<http://www.bortzmeyer.org/xml-to-csv.html>

Issues d'une question posée sur la liste XML-fr <<http://xmlfr.org/listes/>>, voici trois façons possibles de traduire un fichier du format XML vers le format CSV. Une précision, tout d'abord, puisque cet article reçoit beaucoup de visites d'utilisateur d'un moteur de recherche qui ont simplement tapé « convertir du xml en csv » dans ce dernier ; **toutes ces méthodes impliquent de programmer**. Il n'est pas possible de trouver un programme tout fait qui convertisse n'importe quel fichier XML en CSV, car les modèles de données sont trop différents.

Notons d'abord que le format CSV, décrit dans le RFC 4180¹, n'est pas structuré : on ne peut pas mettre des tuples dans d'autres tuples, contrairement à ce que permettent des formats hiérarchiques comme XML ou JSON. D'une manière générale, si un élément XML peut apparaître plusieurs fois, la traduction en CSV va être problématique.

Prenons par exemple de fichier XML des langues parlées au Sénégal (j'ai utilisé les informations publiées par SIL) :

```
<!-- http://www.ethnologue.com/ and ISO 639-2. Le nombre de locuteurs
(très approximatif) est en milliers. -->
<languages>
  <language>
    <code>wo</code>
    <name>Wolof</name>
    <name>Ouolof</name>
    <speakers>3568</speakers>
  </language>
  <language>
    <speakers>607</speakers>
    <name>Manding</name>
    <name>Mandinka</name>
```

1. Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc4180.txt>

```

<code>man</code>
</language>
<language>
  <code>dyo</code>
  <speakers>293</speakers>
  <name>Jola-Fonyi</name>
</language>
<language>
  <code>snk</code><name>Soninke</name><speakers>194</speakers>
</language>
</languages>

```

Si on veut le traduire en CSV, on devra choisir comment placer les différents noms, chaque langue pouvant en avoir plusieurs. Une solution courante est de les mettre dans une seule case, séparé par un autre séparateur (par exemple un point-virgule si le séparateur principal est une virgule).

Ici, nous allons simplifier le problème en ne mettant qu'un nom par langue :

```

<!-- http://www.ethnologue.com/ and ISO 639-2. Le nombre de locuteurs
(très approximatif) est en milliers. -->
<languages>
  <language>
    <code>wo</code>
    <name>Wolof</name>
    <speakers>3568</speakers>
  </language>
  <language>
    <speakers>607</speakers>
    <name>Manding</name>
    <code>man</code>
  </language>
  <language>
    <code>dyo</code>
    <speakers>293</speakers>
    <name>Jola-Fonyi</name>
  </language>
  <language>
    <code>snk</code><name>Soninke</name><speakers>194</speakers>
  </language>
</languages>

```

La première technique employée sera XSLT, un langage déclaratif. Notre script sera :

```

<!DOCTYPE stylesheet [
<!ENTITY newln "&#xA;">
]>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version='1.0'>

  <xsl:output method="text" encoding="utf-8"/>

  <xsl:template match="languages">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="language">
    <xsl:apply-templates select="code"/>
    <xsl:text>,</xsl:text>

```

```

    <xsl:apply-templates select="name"/>
    <xsl:text>,</xsl:text>
    <xsl:apply-templates select="speakers"/>
    <xsl:text>&newln;</xsl:text>
</xsl:template>

<xsl:template match="name">
    <xsl:value-of select="text()" />
</xsl:template>

<xsl:template match="speakers">
    <xsl:value-of select="text()" />
</xsl:template>

<xsl:template match="code">
    <xsl:value-of select="text()" />
</xsl:template>

<xsl:template match="text()">
    <!-- Ignore everything else -->
</xsl:template>

</xsl:stylesheet>

```

Et son utilisation, ici avec xsltproc <<http://xmlsoft.org/XSLT/>> :

```
% xsltproc -o langues.csv langues2csv.xslt langues.xml
```

produira :

```

wo,Wolof,3568
man,Manding,607
dyo,Jola-Fonyi,293
snk,Soninke,194

```

Tout le monde n'aime pas XSLT. La deuxième méthode va donc utiliser un langage impératif, Python, avec le module ElementTree <<http://effbot.org/zone/element-index.htm>> :

```

#!/usr/bin/python

import cElementTree
import sys

if len(sys.argv) <= 1:
    raise Exception("Usage: %s xml-file" % sys.argv[0])

filename = sys.argv[1]
tree = cElementTree.fromstring(open(filename).read())
lang_elements = tree.getiterator("language")
for lang in lang_elements:
    for characteristic in lang:
        if characteristic.tag == "code":
            code = characteristic.text
        elif characteristic.tag == "name":
            name = characteristic.text
        elif characteristic.tag == "speakers":
            speakers = int(characteristic.text) * 1000
    print "%s,%s,%i" % (code, name, speakers)

```

<http://www.bortzmeyer.org/xml-to-csv.html>

Et il donnera quasiment le même fichier CSV (à part qu'on a traduit les milliers en unités).

Et si on préfère les langages fonctionnels, voici un exemple en Haskell, en utilisant la bibliothèque `HaXml` [<http://www.cs.york.ac.uk/fp/HaXml/>](http://www.cs.york.ac.uk/fp/HaXml/) :

```
import Text.XML.HaXml
import System
import IO

rootOf :: Document -> Element
rootOf (Document _ _ r _) = r

nameOf :: Element -> Name
nameOf (Elem n _ _) = n

contentsOf :: Element -> [Content]
contentsOf (Elem _ _ cs) = cs

textOf :: Element -> String
textOf (Elem _ _ cs) = concat (map show cs)

instance Show Content where
    show (CString _ value) = value
    show (CElem e) = nameOf e
    show _ = "Undefined"

showVal :: Content -> String
showVal (CElem e) = textOf e

firstChild :: Name -> Content -> Content
firstChild tagname item = head (concat (map (tag tagname) (children item)))

formatCSV :: Content -> String
formatCSV l =
    let code = firstChild "code" l in
    let name = firstChild "name" l in
    let speakers = firstChild "speakers" l in
    showVal code ++ "," ++ showVal name ++ "," ++ showVal speakers

main = do
    myargs <- getArgs
    if (length myargs) == 0 then
        error "Usage: lang2csv xml-file"
    else
        putStr ""
    let filename = head myargs
        f <- IO.openFile (filename) IO.ReadMode
        input <- IO.hGetContents f
        let xmltree = rootOf (xmlParse filename input)
            languages = concat (map (tag "language") (contentsOf xmltree))
        mapM putStrLn (map formatCSV languages)
```

Si le fichier XML contient des caractères ennuyeux comme des vraies virgules ou comme des sauts de ligne, il faut prendre d'avantage de précautions.

Pour les virgules, Alain Couthures conseille :

- D'utiliser le point-virgule comme séparateur, moins fréquent dans les données et que beaucoup de logiciels (comme Excel) acceptent.
- D'échapper les vraies virgules. Il n'existe pas de norme pour cela mais encadrer les virgules par des guillemets semble fonctionner avec ledit Excel.

– Sinon, de remplacer les virgules, par exemple par des points (Excel, encore lui, accepte en entrée un chiffre avec un point décimal, même dans sa version française). En XSLT, un test avec `contains()` permet de détecter la présence du séparateur dans la donnée et un appel à `translate()` permet de changer des caractères en autres, comme par exemple tous les " " en " ".
 Pour les autres, le mieux est d'avoir des données en XML normalisées, conformes à un schéma strict qui interdit les espaces, les sauts de ligne, etc. C'est plus facile à dire qu'à faire : si on utilise W3C Schema ou bien RelaxNG avec la bibliothèques de types de ces W3C Schemas, on se heurte à une limite (décrite en `<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/#rf-whiteSpace>`). Ces types sont normalisés avant les tests des motifs et il n'y a donc pas moyen d'éliminer les espaces de début et de fin. Un seul type échappe à cette règle, `string` et le schéma RelaxNG ci-dessous l'utilise avec succès :

```
# XML files that follow this schema do not create problems when
# converting to CSV
start = element languages {lang+}

lang = element language {code & name+ & speakers}

code = element code {xsd:string { minLength = "2" maxLength = "3"
                                pattern = "[a-z]+" }}

name = element name {xsd:string { pattern="\S.*\S" }}

# You have to use xsd:string and not xsd:integer because integer is
# normalized so the conditions always succeed.
speakers = element speakers {xsd:string {pattern="[0-9]+"}}
```

Un test avec `rnv <http://www.davidashen.net/rnv.html>` détecte les problèmes facilement. Ici, ligne 8, j'ai écrit :

```
<name>
Wolof</name>
```

et `rnv` le détecte :

```
% rnv schema.rnc languages.xml
languages.xml
languages.xml:8:5: error: invalid data or text not allowed
required:
    data http://www.w3.org/2001/XMLSchema-datatypes^string
```

Merci à Eric van der Vlist pour son aide inestimable sur ce dernier point.

On ne peut pas toujours forcer les fichiers qu'on veut convertir à être conforme à un schéma. Souvent, ces fichiers arrivent comme ils sont et on n'a pas la possibilité de réclamer. Dans ce cas, c'est le programme de conversion qui doit se charger d'éliminer ces espaces et sauts de ligne gênants. En XSLT, la fonction `normalize-space` est très pratique. Si je change mes "templates" XSLT ainsi :

```
<xsl:template match="name">
  <xsl:value-of select="normalize-space(text())"/>
</xsl:template>
```

plus de problèmes, même si le fichier XML contient des espaces ou des sauts de ligne.