XSLT (eXtensible Stylesheet Language Transformation)

Table des matières

1 Introduction

1.1 Définition de XSL

• XSL (*eXtensible Stylesheet Language*) c'est
- XSLT (*XSL Transformation*),
- FO (*Formatting Object*)

• XSLT est un langage permettant de produire un document XML ou texte à partir d'un autre document par application de règles de transformation.

C'est donc un outil essentiel dans une chaîne de traitement XML.

• FO est une DTD XML qui définie la présentation d'un texte sur un document papier (PS, GV, PDF, DPS).

1.2 Structure d'une Feuille de style

```
<?xml version="1.0" encoding="iso-8859-1" ?>

<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0"
   id="identifiant"                              (?)
   extension-element-prefixes="liste"            (?)
   exclude-result-prefixes="liste"               (?)
   >

   ... déclarations optionnelles ...             (?)
   ... règles de transformation ...

</xsl:stylesheet>
```

Une feuille de style XSL est un document XML (version 2.0 en préparation).

Le symbole (?) indique le caractère optionnel d'un attribut ou d'un élément.

2 Les déclarations

2.1 Importation de feuilles XSL

• Importation de feuilles XSL avec gestion de la priorité :

```
<xsl:import href="URL_de_la_feuille_XSL" />
```

Les règles importées sont moins prioritaires que les règles définies dans la feuille courante.

Cette déclaration doit figurer en tête d'une feuille de style.

2.2 Inclusion de feuille XSL

• Inclusion de feuilles XSL (aucun impact sur la priorité) :

```
<xsl:include href="URL_de_la_feuille_XSL" />
```

2.3 Format de sortie

• Déclaration optionnelle du format de sortie :

```
<xsl:output
   method="xml|html|text|... (xml)"            (!)
   version="nu_de_version (1.0)"               (?)
   encoding="encodage (UTF-8)"                 (?)
   indent="yes|no (yes)"                       (?)
   xml-declaration="yes|no (yes)"              (?)
   doctype-system="URL_de_la_DTD"              (?)
   doctype-public="FPI_de_la_DTD"              (?)
   cdata-section-elements="liste d'éléments"   (?)
   ...
   />
```

Attention, l'encodage par défaut est UTF-8 donc illisible pour certains éditeurs de texte.

La méthode de sortie par défaut est XML.

2.4 Gestion des espaces

• Politique par défaut de gestion des blancs pour les noeuds de type texte.

```
<xsl:preserve-space
   elements="liste_d'éléments"             (!)
   />
```

```
<xsl:strip-space
   elements="liste_d'éléments"             (!)
   />
```

2.5 Déclaration des paramètres

• Il est possible d'ajouter des paramètres à une feuille XSL afin de changer son fonctionnement :

```
<xsl:param name="debug">false</xsl:param>

<xsl:param
   name="nom"                                  (!)
   select="expression-XPTAH"                   (!)
   />
```

3 Les règles de transformation

3.1 Présentation

Déclaration d'une règle :

```
<xsl:template
   match="pattern_XPATH"                       (!)
   mode="mode ()"                              (?)
   priority="nombre (0)"                       (?)
   >

   ... élément(s) de remplacement ...          (?)

</xsl:template>
```

Le *pattern XPATH* est une ou plusieurs expressions XPATH connectées par un **ou** « | ».

```
<xsl:template
    match="section/nom|chapitre/titre|def">
    <p>nom : <xsl:value-of select="."/></p>
</xsl:template>
```

• Les éléments de remplacement utilisent la clause ci-dessous pour traiter les fils de l'élément courant.

```
<xsl:apply-templates/>
```

• L'attribut `mode` attache la règle au mode en question.

```
<xsl:template match="section" mode="noir-et-blanc">
    <!-- Mise en forme d'une section en N & B -->
...
</xsl:template>

<xsl:template match="section">
    <!-- Mise en forme d'une section en couleur -->
...
</xsl:template>
```

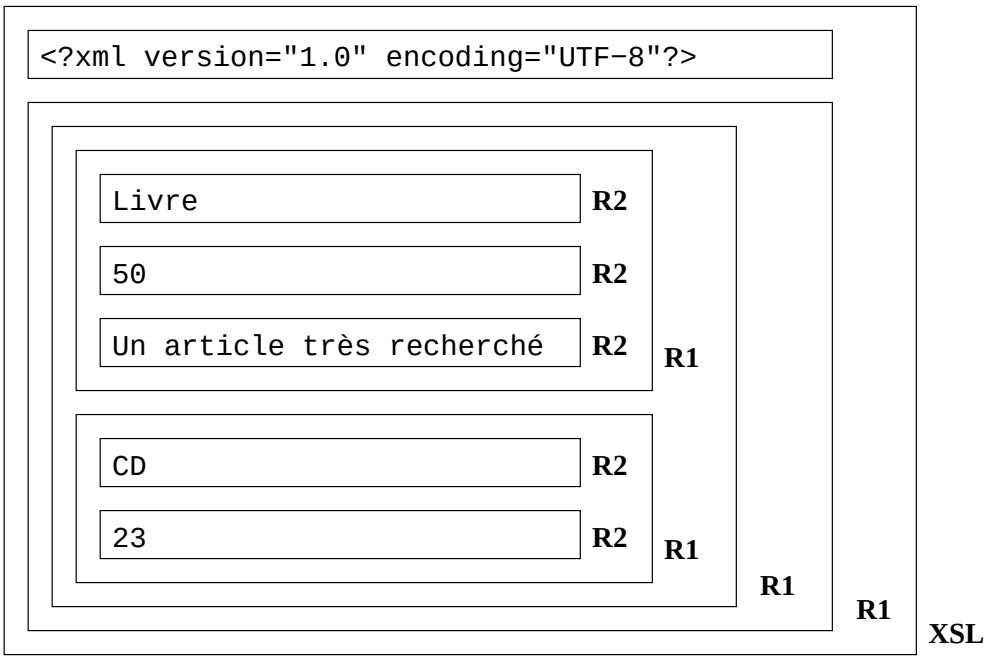
3.2 Traitement de l'exemple

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE stock SYSTEM "essai.dtd">
<stock>
  <produit>
    <nom> Livre </nom><prix monnaie="Francs"> 50 </prix>
    <comment> Un article très recherché </comment>
  </produit>
  <produit>
    <nom> CD </nom><prix monnaie="Euros"> 23 </prix>
  </produit>
</stock>
```

La feuille de style vide contient deux règles par défaut :

```
<xsl:template match="*|/">       <!-- R1 -->
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="text()">    <!-- R2 -->
  <xsl:value-of select="."/>
</xsl:template>
```

L'application d'une feuille de style vide donne :

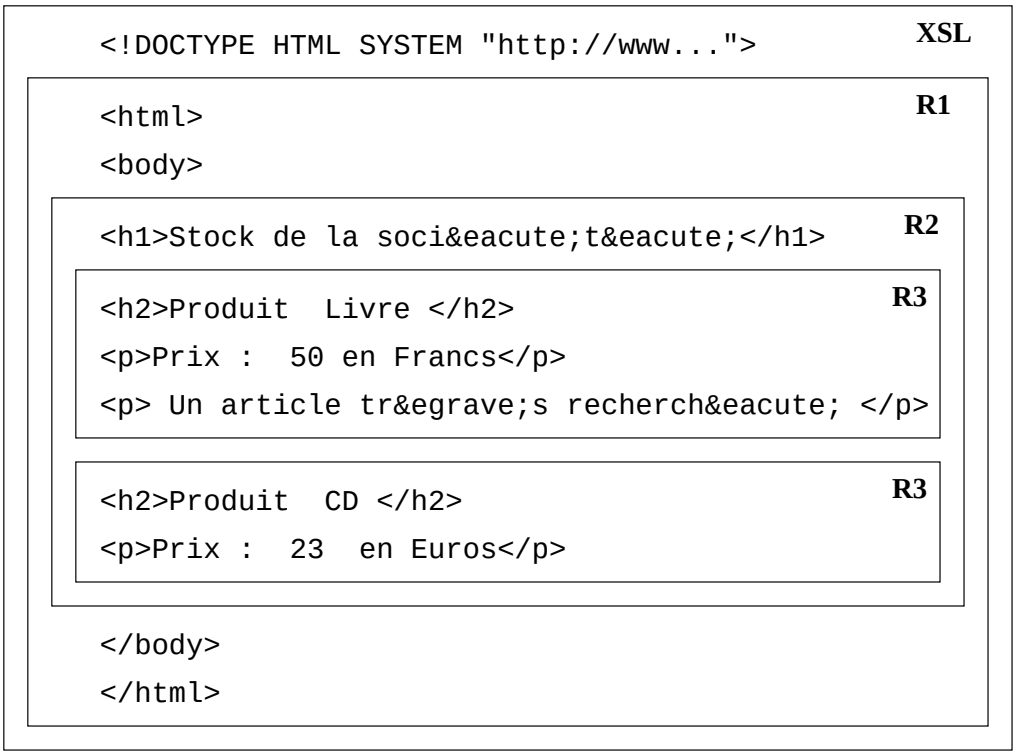


Le texte est codé en UTF-8.

3.3 Exemple de feuille de style

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:output
  method="html"
  doctype-system="http://www.w3.org/TR/REC-html40/loose.dtd"/>

<xsl:template match="/">                          <!-- R1 -->
    <html><body>
    <xsl:apply-templates/>
    </body></html>
</xsl:template>

<xsl:template match="stock">                      <!-- R2 -->
    <h1>Stock de la société</h1>
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="produit">                    <!-- R3 -->
    <h2>Produit <xsl:value-of select="nom"/></h2>
    <p>Prix : <xsl:value-of select="prix"/>
    en <xsl:value-of select="prix/@monnaie"/></p>
    <xsl:if test="comment">
        <p><xsl:value-of select="comment"/></p>
    </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

L'application de la feuille de style précédente donne :



3.4 Traitement des sous-éléments

```
<xsl:apply-templates
   select="pattern_XPATH (*)"                    (?)
   mode="mode ()"                                (?)
   />
```

L'attribut `select` permet de choisir les sous-éléments à traiter. Il permet de ne pas suivre l'ordre du document XML.

L'attribut `mode` change le mode par défaut. On selectionne donc un nouveau jeu de règles.

4 Construction de l'arbre résultat

4.1 Noeud textuel par XPATH

• Produire un noeud textuel identifié par une expression XPATH :

```
<xsl:value-of
   select="expression_XPATH"                    (!)
   disable-output-escaping="no|yes (no)"        (?)
   />
```

Exemples :

```
<xsl:value-of
   select="/stock/produit[nom='Livre']/prix"
   />

<xsl:value-of
   select="//prix[@monnaie='Euros']"
   />
```

Seul le premier arbre est renvoyé.

4.2 Noeud textuel constant

• Insérer un fragment de texte :

```
<xsl:text
   disable-output-escaping="no|yes (no)"        (?)
   >
   ... texte à insérer ...
</xsl:text>
```

Exemples :

```
<xsl:text>
   Vive &amp;nbsp;</xsl:text>

<xsl:text
   disable-output-escaping="yes">
   Vive &amp;nbsp;</xsl:text>
```

Pour obtenir « `Vive  ` ».

4.3 Noeud élément et attribut

• Pour insérer un noeud de type élément dont on connait le nom il suffit de le placer dans la règle :

```
<mon-element message="bonjour">
   <xsl:value-of select="//message"/>
</mon-element>
```

• Une expression XPATH peut être utilisée dans la valeur d'un attribut si elle est entourée d'accolades.

```
XML   |  <image fichier="maison.jpg"/>

XSL   |  <xsl:template match="image">
      |    <img
      |      src="{@fichier}"
      |      alt="Fichier {@fichier}"/>
      |  </xsl:template>

HTML  |  <img src="maison.jpg"
      |    alt="Fichier maison.jpg"/>
```

4.4 Noeud élément et attribut dynamique

• Quand on ne connait pas le nom de l'élément à insérer (idem pour un attribut), on utilise :

```
<xsl:element
   name="nom"                                    (!)
   namespace="espace_de_nom ()"                  (?)
   use-attribute-sets="nom-de-liste"             (?)
   >
   ... contenu de l'élément ...
</xsl:element>
```

• Même chose pour un attribut :

```
<xsl:attribute
   name="nom"                                    (!)
   namespace="espace_de_nom ()"                  (?)
   >
   ... valeur de l'attribut ...
</xsl:attribute>
```

Exemple : effectuer la transformation suivante

```
<prix monnaie="Euros">
  10                        ==>   <Euros prix="10"/>
</prix>
```

avec la règle

```
<xsl:template match="prix">
   <xsl:element name="{@monnaie}">
      <xsl:attribute name="prix">
         <xsl:value-of select="."/>
      </xsl:attribute>
   </xsl:element>
</xsl:template>
```

Pour terminer, notons que cette notation

```
<mon-element id="{un-expression-xpath}">
   ... définition du contenu ...
</mon-element>
```

est équivalente à

```
<mon-element>
   <xsl:attribute name="id">
      <xsl:value-of select="un-expression-xpath"/>
   </xsl:attribute>
   ... définition du contenu ...
</mon-element>
```

On peut donc commencer par définir un élément, puis lui ajouter des attributs.

4.5 Copie de noeud sans attribut

• Copie le noeud courant sans les attributs :

```
<xsl:copy
   use-attribute-sets="nom-de-liste"         (?)
   >
   ... contenu de l'élément copié ...
</xsl:copy>
```

Exemple : tout recopier en traversant l'arbre.

```
<xsl:template match="*">
   <xsl:copy>
      <xsl:apply-templates select="@*"/>
      <xsl:apply-templates/>
   </xsl:copy>
</xsl:template>


<xsl:template match="@*">
   <xsl:attribute name="{name(.)}">
      <xsl:value-of select="."/>
   </xsl:attribute>
</xsl:template>
```

4.6 Copie de noeuds

• Copie directe d'un ensemble de noeuds :

```
<xsl:copy-of select="pattern_XPATH"/>
```

Exemple : ajouter la quantité dans chaque produit :

```
<xsl:template match="produit">
   <produit>
      <xsl:copy-of select="nom|prix|comment"/>
      <quantité>inconnue</quantité>
   </produit>
</xsl:template>
```

4.7 Liste d'attributs réutilisable

Forme générale :

```
<xsl:attribute-set
   name="nom-de-liste"                          (!)
   use-attribute-sets="nom-de-liste"            (?)
   >

   <xsl:attribute name="nom1">valeur1</xsl:attribute>
   <xsl:attribute name="nom2">valeur2</xsl:attribute>
   ...
</xsl:attribute-set>
```

Utilité : regrouper les définitions d'attributs pour les réutiliser associées à plusieurs éléments (tableaux, paragraphes, images, etc.)

Les listes sont définis en dehors des règles.

4.8 Noeud commentaire

• Insérer un commentaire :

```
<xsl:comment>
   ... texte du commentaire ...
</xsl:comment>
```

4.9 Noeud instruction de traitement

L'instruction XSL

```
<xsl:processing-instruction
   name="cible"                                 (!)
   >

   arg1='val1'
   arg2='val2'

</xsl:processing-instruction>
```

donne dans le document produit

```
<?cible arg1='val1' arg2='val2' ?>
```

5 Les éléments de structure

5.1 Les sections conditionnelles

• Le test est vrai ssi l'expression XPATH renvoie un ensemble non vide de noeuds :

```
<xsl:if test="pattern_XPATH">
   ...
</xsl:if>
```

• Le choix multiple

```
<xsl:choose>
   <xsl:when test="pattern_XPATH_1">
      ...
   </xsl:when>
   <xsl:when test="pattern_XPATH_2">
      ...
   </xsl:when>
   ...
   <xsl:otherwise>
      ...
   </xsl:otherwise>
</xsl:choose>
```

5.2 Boucle sur des ensembles de noeuds

• L'instruction XSL `for-each` modifie le noeud courant :

```
<xsl:for-each select="pattern_XPATH">
   ...
</xsl:for-each>
```

Exemple

```
<xsl:template match="/">
    <h1>Liste des noms</h1>
    <xsl:for-each select="//nom">
        <p>nom : <xsl:value-of select="."/></p>
    </xsl:for-each>
</xsl:template>
```

5.3 Tri d'un ensemble de noeuds

• L'instruction `sort` associée à `for-each` modifie l'ordre des noeuds :

```
<xsl:sort
   select="pattern_XPATH (.)"                              (?)
   data-type="text|number (text)"                          (?)
   order="ascending|descending (ascending)"                (?)
   case-order="upper-first|lower-first (upper-first)"       (?)
   lang="nom_de_langue (langue système)"                   (?)
   />
```

Exemple

```
<xsl:template match="/">
    <h1>Liste des produits</h1>
    <xsl:for-each select="//produit">
        <xsl:sort select="nom"/>
        <p>nom : <xsl:value-of select="nom"/></p>
        <p>prix : <xsl:value-of select="prix"/></p>
    </xsl:for-each>
</xsl:template>
```

6 Les variables XSL

6.1 Définition de variables

• Les variables contenants un fragment de l'arbre resultat :

```
<xsl:variable name="nom_de_la_variable">
   ... contenu ...
</xsl:variable>
```

Les variables peuvent être globales ou locales. Exemple :

```
<xsl:variable name="meta">
   <meta>
      <link rel="stylesheet" type="text/css" href="style.css" />
   </meta>
</xsl:variable>

<xsl:template match="/">
   <html>
     <xsl:copy-of select="$meta"/>
     <body><xsl:apply-templates/></body>
   </html>
</xsl:template>
```

• Les variables contenants un fragment de l'arbre d'entrée :

```
<xsl:variable
   name="nom_de_la_variable"
   select="expression_XPATH"
   />
```

Exemple :

```
<xsl:variable name="produits" select="//produits"/>
<xsl:variable name="pas.cher" select="$produits[prix &lt; 10]"/>
<xsl:variable name="gratuit"  select="$pas.cher[prix = 0]"/>

<xsl:template match="/">
   ...
   <xsl:for-each select="$pas.cher">
      ...
   </xsl:for-each>
   ...
</xsl:template>
```

6.2 Priorité entre les règles

• Les règles internes sont plus prioritaire que les règles importées. La clause `<xsl:apply-imports/>` permet de choisir les règles importées.

Feuille `base.xsl` :

```
<xsl:template match="pre">
   <xsl:copy-of select="."/>
</xsl:template>
```

Feuille `mise-en-forme.xsl` :

```
<xsl:import href="base.xsl" />

<xsl:template match="pre">
   <center><xsl:apply-imports /></center>
</xsl:template>
```

L'application de la feuille `mise-en-forme.xsl` donne :

```
<center><pre>...</pre></center>
```

7 Règles nommées et paramètres

7.1 Définition des règles nommées

Il est possible d'appeler une règle par son nom :

```
<xsl:template name="liens">
   <xsl:param name="href"   />
   <xsl:param name="target"  >_blank</xsl:param>

   <a href="{$href}" target="{$target}">
     <xsl:apply-templates/>
   </a>
</xsl:template>
```

Les paramètres se définissent et s'utilisent comme des variables :

```
<xsl:call-template name="liens">
   <xsl:with-param name="target">
      ALL
   </xsl:with-param>
   <xsl:with-param name="href" select="'ma-page.html'" />
</xsl:call-template>
```

7.2 Paramètres d'une feuille XSL

Il est possible d'ajouter des paramètres à une feuille XSL afin de changer son fonctionnement :

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">

<xsl:output method="html"/>

<xsl:param name="debug">false</xsl:param>
```

Les paramètres se définissent et s'utilisent comme des variables :

```
<xsl:if test="$debug = 'true'">
   MODE DEBUG
</xsl:if>
<xsl:if test="$debug = 'false'">
   MODE NON DEBUG
</xsl:if>
```

8 Traitement des espaces de noms

Il est possible de travailler sur plusieurs espaces de noms dans une même feuille XSL :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<stock xmlns="http://www.luminy.univ-mrs.fr">
   ...
</stock>
```

La feuille de style :

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:lum="http://www.luminy.univ-mrs.fr"
    xmlns:dil="http://www.dil.univ-mrs.fr"
    version="1.0">

<xsl:template match="/">
    <dil:prix>
    <xsl:for-each select="//lum:prix">
        <dil:monprix valeur="{.}"/>
    </xsl:for-each>
    </dil:prix>
</xsl:template>
```

L'application de la feuille de style donne :

```
<?xml version="1.0" encoding="UTF-8"?>
<dil:prix
    xmlns:dil="http://www.dil.univ-mrs.fr"
    xmlns:lum="http://www.luminy.univ-mrs.fr">
<dil:monprix valeur="50"/>
<dil:monprix valeur="23"/>
</dil:prix>
```

Si `prix` n'est plus dans l'espace `dil` nous obtenons :

```
<?xml version="1.0" encoding="UTF-8"?>
<prix
    xmlns:dil="http://www.dil.univ-mrs.fr"
    xmlns:lum="http://www.luminy.univ-mrs.fr">
<dil:monprix valeur="50"/>
<dil:monprix valeur="23"/>
</prix>
```

9 Indexation et utilitaires

9.1 Indexation du document XML

• Création de clés :

```
<programme>
   <module>
      <cours>
         <nom>XML</nom>
         <enseignant>Jean-Luc Massat</enseignant>
...
```

```
<xsl:key
   name="cours"                                    (!)
   match="/programmes/module/cours"                (!)
   use="nom"                                       (!)
   />
```

Il est possible de faire référence à un noeud `cours` à partir de son nom dans une expression XPATH :

```
Enseignant d'XML :
<xsl:value-of
   select="key('cours', 'XML')/enseignant"/>
```

9.2 Utilitaires

Émission d'un message d'erreur

```
<xsl:message
   terminate="yes|no"                              (?)
   />
   ... texte du message ...
</xsl:message>
```

Pour imprimer un message et éventuellement arrêter le processeur XSL.

10 Extension d'XPATH par XSL

XPATH est entendu :

- ajout de la fonction `generate-id(noeud)` qui renvoie un identifiant unique :

```
<!-- test de noeuds identiques -->
<xsl:if test="generate-id($n1) = generate-id($n2)">
   ...
</xsl:if>
```

- ajout de la fonction `current()` qui renvoie le noeud courant

```
<xsl:template ...>
   <xsl:variable name="p" select="//article[@id = current()/@id]"/>
   ...
</xsl:template ...>
```

- ajout de la fonction `key(index,clef)`,
- ajout du **ou** dans les expressions.

- ajout de la fonction `format-number(nombre,format)` qui renvoie une chaîne issue du formatage d'un nombre :

```
<xsl:value-of select='format-number(500100, "###,###.00")' />
```

- ajout de la fonction `document(URI)` qui renvoie le document XML identifié par l'URI

```
<xsl:value-of select="document('stock.xml')//produit[prix > 10]" />
```

11 Étendre XSLT

11.1 Étendre XSL avec JAVA

Il est possible, dans une feuille XSL, de faire appel à des routines externes (écrites en Java, Perl, Javascript, TCL, C, etc.)

Avec la classe Java

```
import java.util.Date;
import java.util.Calendar;

public class GetDate
{
  public static Date getDate(String full)
    {
    Date d = Calendar.getInstance().getTime();
    if (! full.equals("true"))
      { d.setMinutes(0); d.setHours(0);}
    return d;
    }
}
```

11.2 Utiliser son extension

et la feuille de style :

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns:java="http://xml.apache.org/xslt/java"
    exclude-result-prefixes="java">

<xsl:output method="html"/>

<xsl:template match="/">
   <xsl:value-of
      select="java:GetDate.getDate('true')" />
   <xsl:value-of
      select="java:GetDate.getDate('false')" />
</xsl:template>
```

nous obtenons

```
Tue Jan 21 17:48:03 CET 2003
Tue Jan 21 00:00:03 CET 2003
```

11.3 Extension standard pour XSLT

• Vous pouvez étudier la librairie EXSLT :

http://www.exslt.org/