Production de documents avec XSLT

Sommaire

- La balise `xsl:output` $\Rightarrow$ comment choisir le format de sortie
- Création de nœuds dans un document $\Rightarrow$ créer des *éléments* et *attributs*
- Création de documents hypertextes $\Rightarrow$ gestion des liens internes et externes
- Plusieurs documents source $\Rightarrow$ la fonction *document*()
- Plusieurs documents résultats $\Rightarrow$ l'élément `xsl:document` (`redirect` dans Xalan)

Choix du format de sortie

La balise `xsl:output`

L'élément `xsl:output` peut être placé au début du programme XSLT pour indiquer le format.

```
<output method='format'>
```

- Le format `xml` (le défaut) indique une sortie dans un dialecte XML (par exemple WML)
- Le format `html` indique une sortie HTML.
- Le format `text` produit un document sans balise.

À propos de HTML et XHTML

Les navigateurs connaissent (au mieux...) HTML 4.0

```
<html>
 <head>
  <title>Exemple de Page HTML 4.0</title>
 </head>
 <BODY>
  <P>Paragraphe 1
  <br><img SRC="image.gif" width=30pt ismap>
  <p>Paragraphe 2
 </body>
</html>
```

Ce n’est pas du XML !

XHTML, version XML de HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitiona
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-t
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de Page XHTML</title>
  </head>
  <body>
     <p>Paragraphe 1</p>
     <p><img src="image.gif"
              alt="une image" width="30pt"
              ismap="ismap"/>
     </p>
  </body>
</html>
```

XSLT et XHTML

Programme XSLT = document XML. On ne peut pas y placer du HTML 4.0, avec des balises non fermées, des attributs sans valeur, etc.
Exemples :

```
<xsl:template match='LOGO'>
  <br><img SRC="image.gif" width=30pt ismap>
</xsl:template>
```

```
<xsl:template match='LOGO'>
  <br/><img src="image.gif"
       alt="Affiche du Film" width="30pt"
       ismap="ismap"/>
</xsl:template>
```

Exemple : sortie HTML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999

<xsl:output method="html" indent="yes"
                version="4.0"
                encoding="ISO-8859-1"/>

<xsl:template match="/">
<html><head><!-- Titre du film   -->
<title>Alien</title></head>
<body>Texte avec des caractères
accentués
</body></html>
</xsl:template>
</xsl:stylesheet>
```

Le résultat

Noter l'indentation, sans effet sur le rendu du document

```
<html>
<head>
<META http-equiv="Content-Type"
         content="text/html; charset=ISO-8859
<title>Alien</title>
</head>
<body>Texte avec des caract&egrave;res
accentu&eacute;s
</body>
</html>
```

Le résultat avec le format `xml`

Ici, l'indentation peut avoir un effet sur le traitement du document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html>
<head>
<title>Alien</title>
</head>
<body>Texte avec des caractères
accentués
</body>
</html>
```

Dernier exemple : le format `txt`

Pour produire des documents non-XML (RTF, Latex, fichiers ASCII, etc).

```
AlienTexte avec des caractères accentués
```

Production d’éléments et d’attributs

Création d'attributs

Pb : comment faire pour introduire des attributs dans un élément quand :

- on ne connaît pas la valeur de l'attribut ;
- on ne connaît ni la valeur ni le nom !

Deux possibilités :

- construction avec `xsl:attribute`
- attribut « interpolé » : une expression XPath encadrée par { }

Exemple : échange attributs/éléments

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ANNUAIRE>
  <PERSONNE>
    <NOM>Amann</NOM>
    <PRENOM>Bernd</PRENOM>
    <EMAIL>amann@cnam.fr</EMAIL>
  </PERSONNE>
  <PERSONNE>
    <NOM>Rigaux</NOM>
    <PRENOM>Philippe</PRENOM>
    <EMAIL>rigaux@lri.fr</EMAIL>
  </PERSONNE>
</ANNUAIRE>
```

Ajout d'un atribut `id`

L'instruction `xsl:attribute` crée un attribut dans le premier élément qui précède.

```
<xsl:template match="PERSONNE">
<PERSONNE>
    <xsl:attribute name='id'>
      <xsl:value-of select='EMAIL'/>
    </xsl:attribute>

    <!-- Copie de tous les éléments fils -->
    <xsl:for-each select="*">
      <xsl:copy-of select="."/>
    </xsl:for-each>
</PERSONNE>
</xsl:template>
```

Le résultat

```
<?xml version="1.0" encoding="UTF-8"?>
<ANNUAIRE>
  <PERSONNE id="amann@cnam.fr">
   <NOM>Amann</NOM>
   <PRENOM>Bernd</PRENOM>
   <EMAIL>amann@cnam.fr</EMAIL></PERSONNE>
  <PERSONNE id="rigaux@lri.fr">
    <NOM>Rigaux</NOM>
    <PRENOM>Philippe</PRENOM>
    <EMAIL>rigaux@lri.fr</EMAIL>
  </PERSONNE>
</ANNUAIRE>
```

Attribut « interpolé »

Plus simple : on place directement l'expression XPath dans l'attribut.

```
<xsl:template match="PERSONNE">
  <PERSONNE id="{EMAIL}">
    <!-- Copie de tous les éléments fils -->
    <xsl:for-each select="*">
      <xsl:copy-of select="."/>
    </xsl:for-each>
  </PERSONNE>
</xsl:template>
```

Calcul de la valeur et du nom

Exemple : on transforme les éléments en attributs.

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

Le résultat

```
<?xml version="1.0" encoding="UTF-8"?>
<ANNUAIRE>
  <PERSONNE NOM="Amann" PRENOM="Bernd"
            EMAIL="amann@cnam.fr"/>
  <PERSONNE NOM="Rigaux" PRENOM="Philippe"
            EMAIL="rigaux@lri.fr"/>
</ANNUAIRE>
```

Calcul des éléments

Exemple inverse : on transforme les attributs en éléments

```
<xsl:template match="PERSONNE">
  <PERSONNE>
    <xsl:for-each select="@*">
      <xsl:element name='{name()}'>
        <xsl:value-of select='.'/>
      </xsl:element>
    </xsl:for-each>
  </PERSONNE>
</xsl:template>
```

Création de documents hypertextes

Langages hypertextes (HTML, WML)

Documents organisés en graphes, avec des liens

- liens entre documents (plutôt HTML)
- liens internes à un document (plutôt WML)

Problèmes :

- exploiter les liens existant, ou les créer s'ils n'existent pas
- exploiter, ou créer, des liens entre plusieurs documents

Petit exemple

```
<FILM>
  <TITRE>Impitoyable</TITRE><ANNEE>1992</ANN
  <GENRE>Western</GENRE><PAYS>USA</PAYS>
  <MES idMES="20"/>
  <ROLES>
   <ROLE idActeur="20">William Munny</ROLE>
   <ROLE idActeur="21">Little Bill Dagget</R
  </ROLES>
</FILM>

<ARTISTE id="20">
  <NOM>Eastwood</NOM><PNOM>Clint</PNOM>
  <ANNEENAISS>1930</ANNEENAISS>
</ARTISTE>
<ARTISTE id="21">
  <NOM>Hackman</NOM><PNOM>Gene</PNOM>
  <ANNEENAISS>1930</ANNEENAISS>
</ARTISTE>
```

Indexation de nœuds avec `xsl:key`

Quand les nœuds sont identifiés : on peut créer un index avec `xsl:key`

```
<xsl:key name="artistes"
match="/FILMS/ARTISTE" use="@id"/>
```

- L'index s'appelle `artistes`
- La clé d'accès est l'attribut `@id`
- On peut l'interroger avec la fonction *key*()

key('artistes', 20) renvoie le nœud `<ARTISTE>` correspondant à C. Eastwood.

Résolution des liens (Résultat)

Règle pour remplacer l'id d'un artiste par son nom et son prénom

```
<xsl:template match="ROLE">
 <xsl:variable name="acteur"
         select="key('artistes', @idActeur)"/
 <b>
   <xsl:value-of
    select="concat($acteur/PNOM,
                   ' ', $acteur/NOM)"/>
 </b> dans le rôle de
         <xsl:value-of select="."/>
</xsl:template>
```

Création d'identifiants

La fonction *generate-id(N)* attribue un identifiant au nœud N.

```
<xsl:template match="FILM">
  <a name="{generate-id(.)}"/>
  <h2>
    <xsl:value-of
          select="concat(TITRE,', ',
                         ANNEE, ', ', GENRE)"/>
  </h2>
</xsl:template>
```

Résultat avec Xalan

```
<html>
<head>
<title>Des films</title>
</head>
<body bgcolor="white">

<a name="N4"></a>
<h2>Impitoyable, 1992, Western</h2>

<a name="N24"></a>
<h2>Seven, 1995, Policier</h2>

<a name="N44"></a>
<h2>Les pleins pouvoirs, 1997, Policier</h2>

</body>
</html>
```

Création d'un lien ROLE -> FILM

```
<xsl:key name="roles" match="ROLE"
                              use="@idActeur"/>

<xsl:template match="ROLE">
 <b><xsl:value-of
     select="concat($acteur/PNOM,' ',
                    $acteur/NOM)"/>
 </b> le rôle de <xsl:value-of select="."/>

 <br/>Voir également :
 <xsl:for-each select="key('roles', @idActeu
   <a href="#{generate-id(../..)}">
      <xsl:value-of select="../../TITRE"/>
   </a>
 </xsl:for-each>
</xsl:template>
```

Résultat

```
<a name="N4"></a>
<h2>Impitoyable, 1992, Western</h2>
<h3>Avec</h3>

<b>Clint Eastwood</b> William Munny<br>
  Voir également :
    <a href="#N44">Les pleins pouvoirs</a>

<b>Morgan Freeman</b> Ned Logan<br>
   Voir également : <a href="#N24">Seven</a>

<a name="N24"></a>
<h2>Seven, 1995, Policier</h2>
<h3>Avec</h3>

<b>Morgan Freeman</b> Somerset<br>
  Voir également : <a href="#N4">Impitoyable
```

Intégration de plusieurs documents

La fonction *document*() permet d'accéder à un document XML à partir d'un programme XSLT

- elle prend en argument le nom d'un fichier (il peut être calculé par une expression XPath)
- elle renvoie la racine de ce document
- à partir de cette racine on peut développer une expression XPath complète pour naviguer dans le document externe

Exemple : recherche des artistes

On suppose que les artistes sont maintenant dans un document *Artistes.xml*.

```
<xsl:template match="ROLE">
<xsl:variable name="A" select="@idActeur"/>
<xsl:variable name="acteur" select=
"document('Artistes.xml')//ARTISTE[@id=$A]"/
<li>
 <xsl:value-of select=
      "concat($acteur/PNOM,' ',$acteur/NOM)"/
 dans le rôle de <xsl:value-of select="."/>
</li>
</xsl:template>
```

Création de plusieurs fichiers

- **Pourquoi** : pour assurer la cohérence des liens, on veut engendrer tous les fichiers HTML en une seule transformation.
- **Comment** : en donnant la possibilité de rediriger la sortie de certaines règles vers des fichiers.
- Cette fonctionnalité est non normalisée en XSLT 1.0. Deviendra l'élément `xsl:document` en XSLT 1.1

Redirection de sortie avec Xalan

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Tran
  xmlns:redirect="org.apache.xalan.xslt.exte
  extension-element-prefixes="redirect">
  <xsl:template match="/">
     <redirect:write file=’sortie.xml’>
      <RESULTAT>
          Il est 10 heures.
      </RESULTAT>
     </redirect:write>
  </xsl:template>
</xsl:stylesheet>
```

Traitement du texte avec XSLT

Problèmes potentiels

Points à considérer quand on traite du texte :

- Les « espaces »
  Tabulation (#x9), saut de ligne (#xA), un retour chariot (#xD) caractère blanc (#x20).
- Les nœuds qui ne contiennent que des espaces : appelons-les **nœuds blancs** ;
- Les caractères comme « & », « < » ou « > » qui peuvent être représentés littéralement, ou remplacés par une référence

Espaces : ce qu'il faut retenir

Au départ, le document source et le programme sont analysés par un parseur :

- Tous les espaces sont conservés dans le document source, y compris les nœuds blancs...
- Tous les espaces sont conservés dans le document XSLT, sauf les nœuds blancs...

Quelques conséquences surprenantes...

Quelques exemples

Le document à traiter

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<INSTRUMENT>
  <NOM>Piano</NOM>
  <DESCRIPTION>Instrument à clavier et
       à cordes frappées.
   </DESCRIPTION>
   <DIVERS>
     Voir clavecin, orgue, etc
   </DIVERS>
</INSTRUMENT>
```

Un premier programme

On traite tous les nœuds.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Tran
  <xsl:output method="xml"
                  encoding="ISO-8859-1"/>
  <xsl:template match="INSTRUMENT">
   <RES>
       <xsl:apply-templates/>
   </RES>
  </xsl:template>
</xsl:stylesheet>
```

Le résultat du premier programme

On ne les voit pas, mais les nœuds blancs du documents source sont insérés !

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RES>
  Piano
  Instrument à clavier et
       à cordes frappées.

     Voir clavecin, orgue, etc
</RES>
```

Le second programme

On indique explicitement les nœuds à traiter.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Tra
 <xsl:output method="xml"
             encoding="ISO-8859-1"/>
 <xsl:template match="INSTRUMENT">
  <RES>
   <xsl:apply-templates
      select="NOM | DESCRIPTION | DIVERS"/>
  </RES>
 </xsl:template>
</xsl:stylesheet>
```

Le résultat du second programme

- Plus de nœuds blancs !
- Les nœuds blancs du programme sont supprimés aussi (dans tous les cas)
- Les retours chariot **dans un nœud texte** sont conservés.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RES>PianoInstrument à clavier et
        à cordes frappées.

     Voir clavecin, orgue, etc
   </RES>
```

Le troisième programme

Deux instructions dans le programme XSLT.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Trans
 <xsl:output method="xml"
                    encoding="ISO-8859-1"/>
  <xsl:template match="INSTRUMENT">
    <RES>
      <xsl:value-of select="NOM"/>
        <xsl:value-of select="DESCRIPTION"/>
    </RES>
  </xsl:template>
</xsl:stylesheet>
```

Le résultat du troisième programme

Ne jamais oublier que les nœuds blancs du programme sont supprimés dans tous les cas.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RES>PianoInstrument à clavier et
        à cordes frappées.
   </RES>
```

Les seuls retours à la ligne sont ceux qui viennent du docu-

ment source.

Contrôler la production de texte

Principaux moyens :

- L'instruction `xsl:strip-space` permet de désigner les éléments pour lesquels on souhaite supprimer les fils qui sont des nœuds blancs
- L'élément `xsl:text` sert à introduire des nœuds blancs dans un programme XSLT
- La fonction *normalize-space*() permet de supprimer les espaces consécutifs dans un nœud de type **Text**

Exemple avec `xsl:strip-space`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Tra
  <xsl:output method="xml"
              encoding="ISO-8859-1"/>
  <xsl:strip-space elements="INSTRUMENT"/>
  <xsl:template match="INSTRUMENT">
    <RES>
       <xsl:apply-templates/>
    </RES>
  </xsl:template>
</xsl:stylesheet>
```

Le résultat

Tous les nœuds blancs fils de <INSTRUMENT> ont été supprimés.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RES>PianoInstrument à clavier et
        à cordes frappées.
     Voir clavecin, orgue, etc
   </RES>
```

À utiliser avec précaution : certains nœuds blancs sont significatifs.

Utilisation de `xsl:text`

Les caractères entre `<xsl:text>` et `</xsl:text>` sont conservés.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Trans
<xsl:output method="xml" encoding="ISO-8859-
  <xsl:template match="INSTRUMENT">
    <RES>
      <xsl:value-of select="NOM"/><xsl:text>

      </xsl:text>
      <xsl:value-of select="DESCRIPTION"/>
    </RES>
  </xsl:template>
</xsl:stylesheet>
```

Le résultat

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RES>Piano

      Instrument à clavier et
       à cordes frappées.
   </RES>
```

La fonction normalize-space()

Normalisation :

- tous les espaces, quel que soit leur type, sont remplacés par des caractères blancs (`#x20`) ;
- les séquences de plusieurs caractères blancs consécutifs sont réduites à un seul ;
- les blancs au début et à la fin sont supprimés

Exemple

```
 <xsl:template match="INSTRUMENT">
  <RES>
   <xsl:value-of select="NOM"/>
   <xsl:text>
   </xsl:text>
   <xsl:value-of
      select="normalize-space(DESCRIPTION)"/
   <xsl:text>
</xsl:text>
   <xsl:value-of
      select="normalize-space(DIVERS)"/>
  </RES>
 </xsl:template>
```

Le résultat

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RES>
Piano
Instrument à clavier et à cordes frappées.
Voir clavecin, orgue, etc
</RES>
```