XSLT en détail

Les principaux éléments

La première ligne

- Le premier élément XSL que l'on rencontre dans une feuille de style XSLT est l'élément <xsl:stylesheet>
  - C'est l'élément racine de toutes les feuilles XSLT
  - Toujours de la forme suivante :

  <xsl:stylesheet version="1.0" xmlns:xls="http://www.w3.org/XSL/Transform">
- Ensuite on définit assez souvent des modèles (ou templates)

Les modèles

- Les feuilles de style XSLT sont en réalité une collection de modèles
- Un modèle est délimité par la balise <xslt:template>
- 2 éléments importants :
  - La section de l'arbre source à laquelle s'applique le modèle
  - La sortie à placer dans l'arbre résultat
- La partie de l'arbre source est spécifiée dans l'attribut match
- Tout ce qui se situe entre la balise de début et de fin du modèle représente ce qui est produit vers l'arbre résultat

Le noeud contextuel

- Ce qui est utilisé comme attribut match devient le **noeud contextuel** de ce modèle
- Toute expression XPath contenue dans le modèle est relative à ce noeud contextuel

```
<xsl:template match="/client">
    <xsl:value-of select="nom"/>
</xsl:template>
```

- Dans l'exemple, l'expression XPath de l'attribut <xsl:value-of> va sélectionner les éléments nom, enfant de l'élément client, choisis pour ce modèle
- Le noeud contextuel de ce modèle est l'élément client qui se trouve à la racine du document (ou de l'arbre source)

Le modèle par défaut

- Si aucun modèle n'est définit dans le document, XSLT en fournit par défaut

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

- Ce modèle identifie tous les éléments du document, racine comprise, et appelle `<xsl:apply-templates>` qui traite tous les enfants
- Il existe un autre modèle par défaut pour tous les noeuds textes et attributs

```
<xsl:template match="text()|@*/">
    <xsl:value-of select="."/>
</xsl:template>
```

- Ce modèle ajoute la valeur du noeud texte ou de l'attribut à l'arbre résultat

Exercice

- Exercice 1 : Sur le fichier XML de référence du cours 3, appliquez lui une feuille XSLT minimale et visualisez le résultat
  - *Réponse dans le fichier XML-Cours4-Exo1.xsl*

Ordre des opérations XSLT

- Quand plusieurs modèles sont présents dans une feuille, dans quelle ordre sont-ils traités ?
- Le processeur XSL débute en comparant la racine du document au modèle qui lui convient le mieux puis les traite dans l'ordre si plusieurs modèles sont applicables

<xsl:template>

```
<xsl:template match="expression Xpath"
              name="nom modèle"
              priority="numero"
              mode="nom du mode" >
```

- L'attribut match permet de sélectionner des noeuds de l'arbre source
- `<xsl:template match="nom">`

  `<xsl:template match="nom[.='John']">`

  Tous les noeuds `nom` ayant une valeur différentes de John correspondent au premier modèle alors que si la donnée est John, cela va corresponde au deuxième modèle
- Mais plusieurs modèles peuvent correspondent à un noeud particulier, il est alors possible d'imposer une priorité via l'attribut `priority`
- L'attribut `name` permet de créer un modèle nommé afin de pouvoir l'appelé depuis une feuille de style
- L'attribut `mode` sert lorsque la même section de l'arbre source doit être traitée à plusieurs reprises

<xsl:apply-templates>

```
<xsl:apply-templates match="expression Xpath"
                     mode="nom du mode" >
```

- Permet d'appeler un modèle depuis un autre modèle
- Si l'attribut `select` est spécifié, le résultat de l'expression Xpath est utilisé comme noeud contextuel, sinon le noeud contextuel courant sera utilisé
- L'attibut `mode` fonctionne avec l'attribut mode de l'élément `<xsl:template>`
- Exemple : (Fichier XML-Cours4-Exemple1.xml)

```
<?xml version= "1.0" ?>
<simple>
<nom>Martin</nom>
<nom>David</nom>
</simple>
```

- Affichons la liste des noms dans une page HTML

Utilisons <xsl:apply-templates>

Comparez les résultats (*fichier XML-Cours4-Exemple1.xsl*)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<HTML>
<BODY>
<xsl:apply-templates/>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <xsl:apply-templates/>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="nom">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

Fonctionnement de l'exemple

- Après examen de la racine, application du modèle la concernant
  - Création des éléments `<html>` et `<body>`
  - Exécution de `<xsl:apply-template>` qui va faire une recherche dans l'arbre source d'autres modèles à appliquer
  - Pour chaque élément <nom>, il recherche s'il existe un modèle à appliquer
    - `<xsl:template match="nom">`
  - Dès qu'il ne reste plus de modèle à appliquer, le processeur XSLT termine en fermant dans l'arbre résultat les balises <body> et <html>

<xsl:value-of>

```
<xsl:value-of select="expression Xpath"
 disable-output-escaping="yes ou no" />
```

- Insère le PCDATA de l'élément résultat de l'expression Xpath dans l'arbre résultat
- Exemple :
  - `<xsl:value-of select="." />` : insère le PCDATA du noeud contextuel courant dans l'arbre résultat
  - `<xsl:value-of select="client/@id" />` : insère le texte de l'attribut id de l'élément <client>
- L'attribut `disable-output-escaping= "yes"` impose la sortie des éléments & et < plutôt que les caractères d'échappements & et <
- Par défaut, `disable-output-escaping="no"`
- Exemple : exécutez sur `<nom>&amp;</nom>`
  - `<xsl:value-of select="nom" disable-output-escaping="yes"/>`
  - `<xsl:value-of select="nom" disable-output-escaping="no"/>`

<xsl:output>

```
<xsl:output method="xml ou html ou text ..."
    version="version" encodage="encodage"
  omit-xml-declaration="yes ou no" standalone="yes ou no"
    cdata-section-elements="sections CDATA"
      indent="yes ou no" />
```

- Élément enfant direct de l'élément <xsl:stylesheet>
- Permet de contrôler la façon dont la sortie est créée
- L'attribut method : plusieurs types de sortie peuvent être utilisés, cela dépend du processeur XSLT utilisé
- Si l'élément output est absent, si l'élément racine de l'arbre résultat est <html>, la méthode de sortie par défaut est html, sinon c'est <xml>
- `omit-xml-declaration` dans le cas ou le résultat serait directement inclus par la suite dans un document xml ou il existe donc déjà un déclaration xml
- `indent` permet d'embellir le résultat pour une meilleur lecture

<xsl:element>

```
<xsl:element name="nom element"
use-attribute-sets="noms des ensembles d'attributs">
```

- Permet d'insérer directement des éléments dans l'arbre résultat
- Exemple :
  - `<xsl:element name="nom">Toto</xsl:element>` ajoute dans l'arbre résultat `<nom>Toto</nom>`
  - ```
    <xsl:template match="nom">
    <xsl:element name="{.}">Toto</xsl:element>
    </xsl:template>
    ```
    - ajoute un élément qui porte comme nom la valeur provenant du noeud contextuel
    - Appliqué à `<nom>Martine</nom>`, cela produira `<Martine>Toto</Martine>`
    ```

Application

- A partir du document de gauche, écrire la feuille XSLT produisant le document dont le début est donnée ci-dessous, à droite (*fichiers XML-Cours4-Exo2.xml et XML-Cours4-Exo2-result.xml, solution XML-Cours4-Exo2.xsl*)

```
<?xml version="1.0" encoding="utf-8"?>
<carnet>
<nom prenom="John" prenom2="Martin" famille="Doe"/>
<nom prenom="Smith" prenom2="Axel" famille="Poyot"/>
<nom prenom="Elisabeth" prenom2="Eva"
famille="Quesnel"/>
<nom prenom="Pierre" prenom2="" famille="Tapetone"/>
<nom prenom="Isabelle" prenom2="Martin"
famille="Buisson"/>
</carnet>
```

```
<?xml version="1.0" encoding="utf-8"?>
<carnet>
<nom>
<prenom>John</prenom>
<prenom2>Martin</prenom2>
<famille>Doe</famille>
</nom>
<nom>
<prenom>Smith</prenom>
<prenom2>Axel</prenom2>
<famille>Poyot</famille>
</nom>
<nom>
<prenom>Elisabeth</prenom>
<prenom2>Eva</prenom2>
<famille>Quesnel</famille>
</nom>
...
```

<xsl:attribute> et <xsl:attribute-set>

```
<xsl:attribute name="nom element">
```

- Fonctionne de la même manière que `<xsl:element>`
- `<nom><xsl:attribute name="id">142</xsl:attribute>Martine</nom>` produit `<nom id="142">Martine</nom>`
- L'element `<xsl:attribute>` doit se trouver avant tout PCDATA
  - `<nom>Martine <xsl:attribute name="id">142</xsl:attribute></nom>` est **incorrecte**
- `<xsl:attribute-set>` permet de définir un ensemble d'attributs applicables à des éléments via l'attribut `use-attribute-sets`
  -
```
<xsl:attribute-set name="idtaille">
    <xsl:attribute name="id">213</xsl:attribute>
    <xsl:attribute name="taille">174</xsl:attribute>
</xsl:attribute-set>
```
  - Ceci définit 2 attributs qui pourront être appliqués à tout élément

Exemple

A partir du fichier résultat de l'application précédente, la feuille XSLT suivante ajout les mêmes attributs à tous les éléments nom (*fichiers XML-Cours4-Exemple2.xml et XML-Cours4-Exemple2.xsl*)

```
…
 <xsl:attribute-set name="idtaille">
    <xsl:attribute name="id">213</xsl:attribute>
    <xsl:attribute name="taille">174</xsl:attribute>
  </xsl:attribute-set>

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

  <xsl:template match="nom">
    <xsl:element name="{name()}" use-attribute-sets="idtaille">
        <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>

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

Exercice

- A partir du fichier résultat de l'application précédente (fichier de gauche), écrire la feuille XSLT qui permet de retrouver le document XML de droite (*fichiers XML-Cours4-Exo3.xml et XML-Cours4-Exo3-result.xml, solution XML-Cours4-Exo3.xsl*)

```
<?xml version="1.0" encoding="utf-8"?>
<carnet>
<nom>
<prenom>John</prenom>
<prenom2>Martin</prenom2>
<famille>Doe</famille>
</nom>
<nom>
<prenom>Smith</prenom>
<prenom2>Axel</prenom2>
<famille>Poyot</famille>
</nom>
<nom>
<prenom>Elisabeth</prenom>
<prenom2>Eva</prenom2>
<famille>Quesnel</famille>
</nom>
...
```

```
<?xml version="1.0" encoding="utf-8"?>
<carnet>
<nom prenom="John" prenom2="Martin"
famille="Doe"/>
<nom prenom="Smith" prenom2="Axel"
famille="Poyot"/>
<nom prenom="Elisabeth" prenom2="Eva"
famille="Quesnel"/>
<nom prenom="Pierre" prenom2=""
famille="Tapetone"/>
<nom prenom="Isabelle"
prenom2="Martin" famille="Buisson"/>
</carnet>
```

<xsl:text>

```
<xsl:text disable-output-escaping= "yes ou no">
```

- Permet d'insérer du texte dans l'arbre résultat
- Exemple :

```
<xsl:text disable-output-escaping="yes">6 est &lt; 7
 &amp; 7 > 6</xsl:text>
```

Résultat :
6 est < 7 & 7 > 6

<xsl:if> et <xsl:choose>

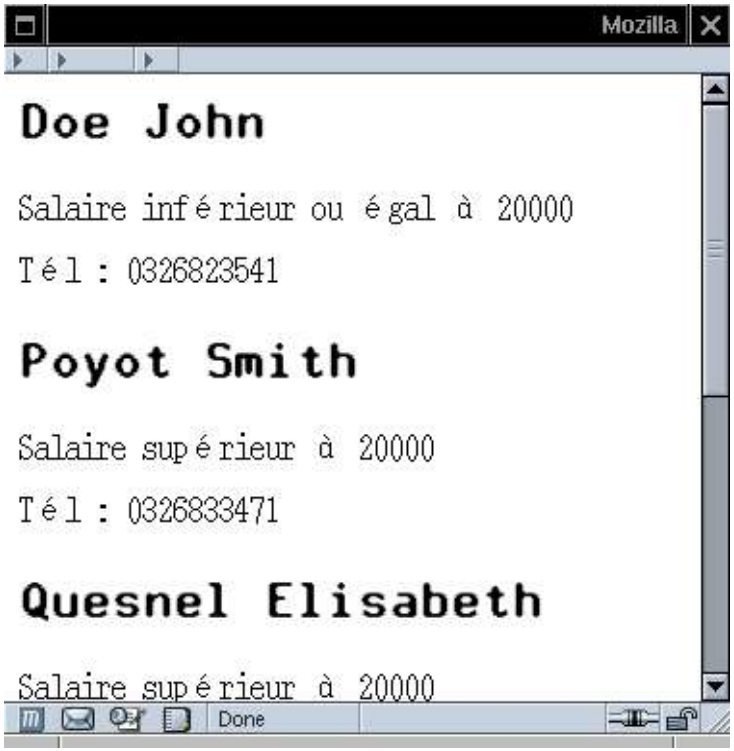
```
<xsl:if test="expression booléenne"></xsf:if>
                    <xsl:choose>
          <xsl:when test="expression booléenne">
          <xsl:when test="expression booléenne">
                     <xsl:otherwise>
                    </xsl:choose>
```

- L'expression booléenne est l'expression XPath convertie en valeur booléenne à l'aide des mêmes règles la fonction `boolean()`
- <xsl:if> ne modifie pas le noeud contextuel comme un match peut le faire dans un template ou un apply-templates

Exercice

- A partir de la source ci-dessous (*fichier XML-Cours4-Exo4.xml*), écrire la feuille XSLT qui permet d'obtenir l'écran de droite (*solution : fichier XML-Cours4-Exo4.xsl*)

```
<?xml version="1.0" encoding="utf-8"?>
<carnet>
<nom>
<prenom>John</prenom>
<prenom2>Martin</prenom2>
<famille>Doe</famille>
<salaire>20000</salaire>
<poste>3541</poste>
<zone>2</zone>
</nom>
<nom>
<prenom>Smith</prenom>
<prenom2>Axel</prenom2>
<famille>Poyot</famille>
<salaire>21250</salaire>
<poste>3471</poste>
<zone>3</zone>
</nom> ...
```



<xsl:for-each>

```
<xsl:for-each select= "expression XPath">
```

- Permet de faire un traitement sur tous les éléments répondants à l'expression Xpath
- <xsl:for-each> est un modèle, il modifie donc le noeud contextuel

```
<xsl:template match= "noms">
      <xsl:for-each select="nom">
      <p><xsl:value-of
   select="prenom"></p>
      </xsl:for-each>
</xsl:template>
```

Le noeud contextuel est nom dans le for-each

prenom est pas nom/prenom

```
<noms>
    <nom>
        <prenom>Toto1</prenom>
    </nom>
    <nom>
        <prenom>Toto2</prenom>
    </nom>
</noms>
```

<xsl:for-each> vs <xsl:template>

- <xsl:for-each> est un modèle, au même titre que <xsl:template>
- Par contre, <xsl:for-each> peut être insérer dans un modèle alors que <xsl:template> doit être autonome

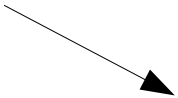
<xsl:copy-of>

```
<xsl:copy-of select="expression XPath">
```

- Permet de prendre des sections de l'arbre source et de les recopier vers l'arbre résultat (*fichiers XML-Cours4-Exemple4.xml et XML-Cours4-Exemple4.xsl, résultat XML-Cours4-Exemple4-result.xml*)

```
<noms>
    <nom>
        <prenom>Toto1</prenom>
    </nom>
    <nom>
        <prenom>Toto2</prenom>
    </nom>
</noms>
```

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



```
<?xml version="1.0" encoding="utf-8"?>

    <nom>
        <prenom>Toto1</prenom>
    </nom>
    <nom>
        <prenom>Toto2</prenom>
    </nom>
```

Exercice

- Nous allons considérer un fichier XML (*XML-Cours4-Exo5.xml*) contenant le nom, prénom et salaire du personnel. Pour chaque nom, un attribut droit de 1 à 2 est défini. En fonction de ce droit nous allons générer un nouveau document XML (*XML-Cours4-Exo5-result.xml*) qui contiendra ou pas le salaire de la personne
- Nous devons retrouver les mêmes informations que l'original pour toutes les personnes de droit 1
- Pour les personnes de droit 2, les salaires ne doivent pas être indiqués
- *Indication* : utilisation des commandes <xsl:copy-of> et l'expression XPath self::salaire

<xsl:copy>

```
<xsl:copy
  use-attribute-sets="noms des attributs de l'ensemble">
```

- Copie simplement le noeud contextuel
- Les enfants et les attributs du noeud contexteul ne sont pas automatiquement copiés dans l'arbre résultat
- Application : Que donne les exemples suivants sur le fichier *XML-Cours4-Exo5.xml* ? (*fichiers XML-Cours4-Exo6-1.xsl et XML-Cours4-Exo6-2.xsl*)

```
<xsl:template match="nom">
    <xsl:copy />
</xsl:template>
```

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

<xsl:sort>

```
<xsl:sort select= "expression Xpath" lang="langue"
          data-type="text ou number"
        order="ascending ou descending"
      case-order="upper-first ou lower-first">
```

- Le tri s'accomplit en ajoutant un ou plusieurs enfants <xsl:sort> à un élément <xsl:apply-templates> ou à un élément <xsl:for-each>
- select choisit l'élément sur lequel vous souhaitez faire le tri
- Si plusieurs éléments <xsl:sort> sont ajoutés, la sortie est triées sur l'élément du premier sort
- data-type signale si l'élément trié est du texte ou des nombres

XSLT en détail

Les autres fonctions

Les modes

- Attribut des éléments <xsl:template> et <xsl:apply-templates>
- Permet d'identifier les mêmes parties de l'arbre source sur lesquels les manipulations à effectuer sont différentes
- Lors de la création d'un modèle, définir le mode
- Pour appeler le mode, utiliser <xsl:apply-templates> en ajoutant l'attribut mode
- Exemple : Réaliser une table des matières en début de document HTML, suivi du corps du document. Dans ce cas, les titres de chaque section vont devoir être affichés de plusieurs façons. Les modes facilitent cela.

Variables, constantes et modèles nommés

```
<xsl:variable name="pi">3,15</xsl:variable>
<xsl:variable name="esp">
          <xsl:text> </xsl:text></xsl:variable>
<xsl:variable name="nom" select="/carnet/nom"/>
```

- Ces variables sont utilisées comme suit :
  - `<xsl:value-of select="$esp"/>`
- Il est possible de faire appelle à une variable dans une autre variable

```
<xsl:variable name="nom">
<xsl:value-of select="nom/prenom"/>
<xsl:value-of select= "$esp"/>
<xsl:value-of select="nom/famille"/>
</xsl:variable>
```

- La valeur de la variable si un attribut select est utilisé est dans ce cas, le résultat de l'expression XPath
- **Attention** : pas de référence circulaire, pas de référence à soi-même
- Garde la notion de variable globale et locale

Modèles nommés

- Permet de faire un appel à un modèle qui peut être appliqué à plusieurs noeuds
- Exemple : mettre en gras une portion de code

```
<xsl:template name="gras">
<B><xsl:value-of select="."/></B>
</xsl:template>
```

- Ces modèles sont appelés par l'élément <xsl:call-template />
- Exemple : mettre en gras des noms

```
<xsl:for-each select="nom">
  <xsl:call-template name="gras"/>
</xsl:for-each>
```