Le langage XSLT

Transformation de fichiers XML

- **Présentation générale**
- Règles élémentaires
- Construction du document résultat
- Structures itératives et conditionnelles
- Variables XSL
- Règles paramétrées, et avec noms
- Autres aspects
- Conclusion

Présentation (1)

- XSL = XML Stylesheet Language
- XSL est un langage de feuilles de styles.
- XSLT-1.0 est une Recommandation du W3C composée de :
  - XSLT: un langage de transformation
  - XPath: déjà étudié, sélection de noeuds
  - XSL-FO: un jeu d'instructions de formatage en XML destiné a la présentation
  -

Présentation (2)

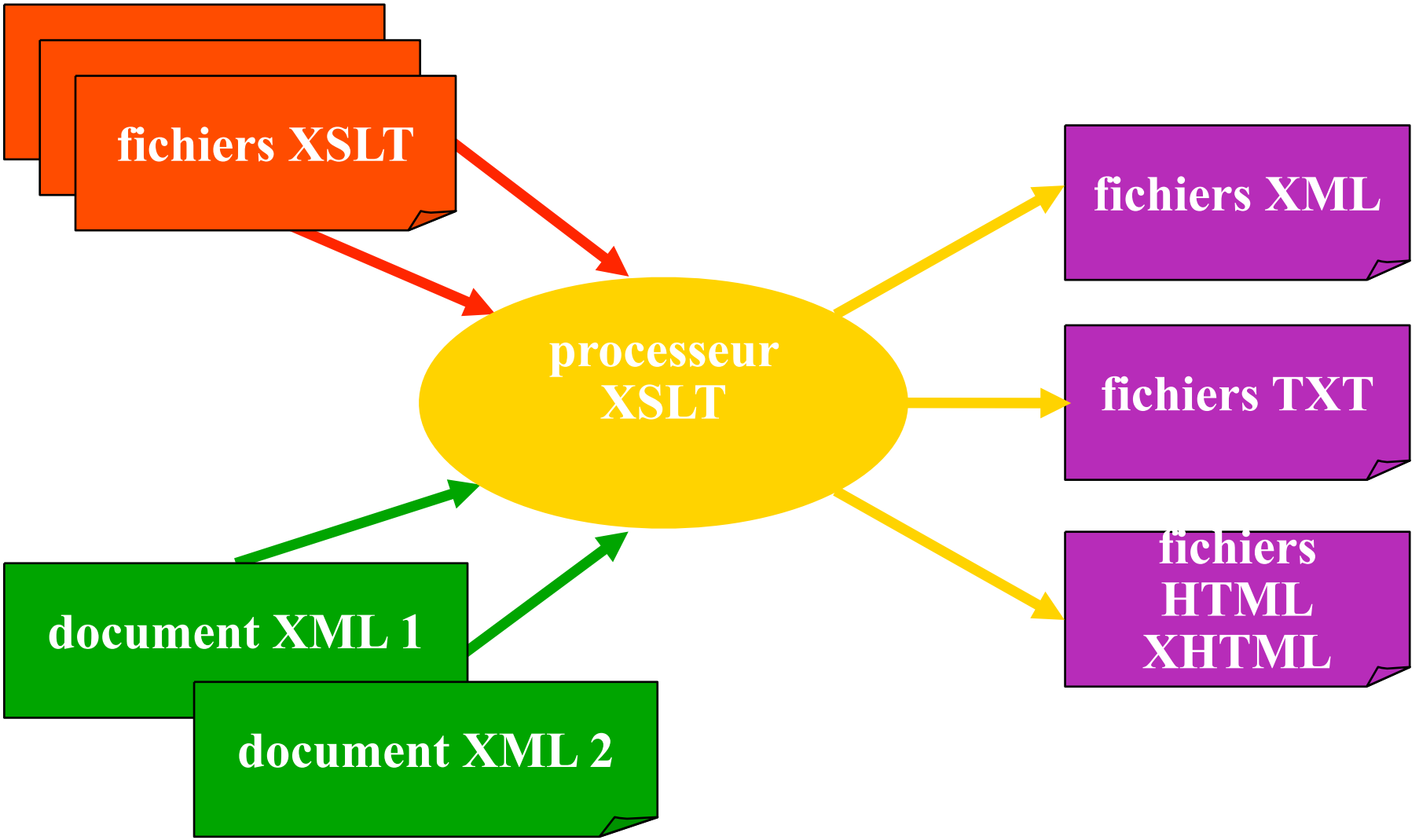
- XSLT permet de transformer un document XML en un autre document XML ou bien dans un autre format, à des fins de:
  - affichage: on peut obtenir un document HTML pour la visualisation (comparable à CSS). On peut également produire des fichiers textes, PDF, …
  - adaptation: on veut adapter les données contenues dans le document pour obéir à une DTD donnée …

XPath, XSL-FO

- **XPath:** XSLT permet de spécifier un certain nombres d'opérations à exécuter lorsqu'on rencontre un élément particulier.
  => On utilise XPath pour désigner les parties du documents sur lesquelles on veut travailler.

- **XSL-FO:** la partie formatage comporte :
  - le modèle de formatage: boites, positionnement, ordonnancement , de quelle manière les pages sont organisées et comment l'information « coule » d'une page à une autre
  - des propriétés d'affichage: text-align, margin-left, background, fontes, …

XSLT = XSL Transformations



Processeur XSLT (1)

- Un évaluateur de feuille de style lit un document XSLT, un document XML et produit un document en sortie en appliquant les instructions/spécifications données dans la feuille de style

- L'évaluateur peut être intégré à un butineur : Firefox, Explorer...

- ou bien intégré au serveur, pour la création dynamique de pages Web, e.g. le projet Apache XML Cocoon: `http://cocoon.apache.org/`

Processeur XSLT

- L'évaluateur peut être un exécutable « standalone » utilisé depuis la ligne de commande. Par exemple en utilisant Xalan :

```
java org.apache.xalan.xslt.Process
-IN file.xml  -XSL file.xsl  -OUT …
```

- Quelques processeurs classiques :
  - Xalan
  - Xerces
  - Saxon
  - Sablotron
  - ...

- Présentation générale
- **Règles élémentaires**
- Construction du document résultat
- Structures itératives et conditionnelles
- Variables XSL
- Règles paramétrées, et avec noms
- Autres aspects
- Conclusion

Structure d'une feuille de style

- Une feuille de style est un document XML de la forme

```
<xsl:stylesheet xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform"
 version="1.0">
    …
   <xsl:template match="path">
       [description du template]
   </xsl:template>
    …
</xsl:stylesheet>
```

- L'espace de noms associé au langage XSLT est
  `http://www.w3.org/1999/XSL/Transform`
- Le préfixe usuel est `xsl`.
- On peut utiliser `xsl:transform` au lieu de `xsl:stylesheet`

Modèle d'exécution

- Une feuille de style XSLT est constituée d'un prélude suivi d'une séquence de règles de transformation de la forme :

```
<xsl:template match="chemin">
        … transformations …
</xsl:template>
```

- Une règle comporte deux parties :
  - un modèle de chemin exprimé en termes du langage XPath (identifie un ensemble de nœuds)
  - une transformation sous la forme d'une séquence de caractères ou d'éléments dont certains sont des instructions XSLT

Modèle d'exécution

- L'évaluation d'un template se fait à partir d'un noeud contexte. Le noeud contexte initial est le nœud racine

- À partir d'un nœud contexte, l'évaluation se déroule de la manière suivante:
  - on cherche le template ayant le modèle de chemin le plus précis et qui matche le nœud contexte
  - on instancie le template retenu à l'étape précédente. La transformation peut créer un fragment du document résultat (en insérant du texte dans le flot de sortie) et instantier de nouveaux templates. L'évaluation continue ensuite récursivement.

- Un ensemble de nœud est évalué en prenant chacun des nœuds de l'ensemble comme nœud contexte puis en concaténant les résultats obtenus.

Règles prédéfinies (= par défaut)

- Les règles prédéfinies s'appliquent en l'absence de règles applicables définies dans la feuille de style. Elles ont une priorité plus faible que celles-ci.

- **Exemple**: pour les nœuds racine et éléments on poursuit le traitement sur les nœuds fils du nœud contexte :

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

- **Exemple**: pour les nœuds textes et attributs on recopie leurs valeurs dans le flot de sortie :

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

Balise template

Cet élément définit un modèle à appliquer à un nœud et à un contexte spécifiques.

Utilisation :

```
<xsl:template name="nomModele"
        match="expressionXPath" mode="nomMode">
 </xsl:template>
```

- name : associe un nom au modèle (optionnel)
- match : indique quel nodeset sera affecté par le modèle (obligatoire)
- mode : permet à un même élément d'avoir plusieurs modèles, selon le contexte (optionnel)
- priority : utilisé en cas de conflit entre deux règles ayant la même condition (optionnel)

Un Premier Exemple

- Application d'une feuille de style XSL à un document XML pour l'afficher dans un navigateur web :

  1/ instruction de traitement dans le fichier XML

  <?xml-stylesheet type="text/xsl" href="wg.xsl"?>

  2/ choix d'un noeud de départ dans la feuille de style

  <xsl:template match="/">

  3/ balise de base

  <xsl:value-of select = "expressionXPath"/>

Un Premier Exemple (2)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <head>  <title>Présentation HTML de wg.xml</title> </head>
    <body>
      <h1>Groupe de travail, <xsl:value-of select="gdt/@month"/>
                                  <xsl:value-of select="gdt/@year" /></h1>
      <h2>Le <xsl:value-of select="gdt/expose/@day"/> <xsl:value-of select="gdt/@month"/> :</h2>
      <p>Présentation de <xsl:value-of select="gdt/expose/speaker" /></p>
      <h3>Titre:</h3>
      <p><xsl:value-of select="gdt/expose/title" /></p>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="wg.xsl"?>
<gdt month="Février" year="2004">
  <expose day="5">
    <speaker>Nicolas Baudru</speaker>
    <title>Netcharts et HMSC</title>
    <time>12h45</time>
    <salle>102</salle>
  </expose>
</gdt>
```

- Présentation générale
- Règles élémentaires
- **Construction du document résultat**
- Structures itératives et conditionnelles
- Variables XSL
- Règles paramétrées, et avec noms
- Autres aspects
- Conclusion

Construction du document résultat

Plusieurs opérations possibles :

- insérer la valeur d'un noeud du document d'entrée
- copier un fragment du document d'entrée
- insérer un nouvel élément de valeur 'statique'
- insérer un nouvel élément de valeur 'dynamique'

Balise xsl:value-of

Cet élément permet d'insérer la valeur d'un nœud dans la transformation.
Ce nœud est sélectionné par une expression XPath.
Cette expression peut correspondre à un élément, à un attribut ou à tout autre nœud contenant une valeur :

<xsl:value-of select="expressionXPath" />

- select : la valeur de l'attribut select est évaluée, et c'est cette évaluation qui sera insérée lors de la transformation. (obligatoire)
- pour un noeud x de type élément, la valeur retournée est la concaténation des noeuds textes du sous-arbre enraciné en x
- pour un ensemble de noeuds, seule la valeur du premier est retournée.

Copie d'un fragment

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

- Recopie dans le flot de sortie le fragment du document à transformer sélectionné par la requête XPath à partir du noeud contexte.

- Ce fragment peut être constitué de plusieurs noeuds.

Exemple :

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

Recopie les éléments expose désignés et leur contenu..

Fragment résultat élément

- `<xsl:element name="balise">...</xsl:element>`

  construit un noeud élément de nom *balise* avec les attributs et contenus définis dans le corps de l'élément xsl:element

Exemple :

```
<xsl:element name="p"><xsl:value-of select="para" /></xsl:element>
```

Crée une balise p contenant le texte présent dans l'élément para du document d'origine.

Fragment résultat attribute

`<xsl:attribute name="att">valeur</xsl:attribute>`

construit un attribut nommé “att”

(à utiliser à l’intérieur d’un élément `xsl:element`)

Exemple :

```
<xsl:element name="image">
   <xsl:attribute name="src">
      <xsl:value-of select=”@adresse”/>
   </xsl:attribute>
   ....
</xsl:element>
```

Crée une balise image, ayant un attribut src dont la valeur est celle de l’attribut adresse de l’élément d’origine :

< image src ="test.gif"></image>

Syntaxe courte

- Dans une feuille de style, on peut en fait insérer directement de nouvelles balises :

  <image> ... </image>

- Pour faire référence au document d'origine, un élément peut contenir dans ses attributs des « expressions attributs » de la forme *{path}*, où *path* est une expression XPath qui est évaluée vers une chaîne de caractères :

  <image src="{@adresse}"></image>

- Inconvénients de la syntaxe courte ?
  - dure à relire si les expressions XPath sont longues
  - automatisation plus complexe

Syntaxe courte/Syntaxe longue ?

On peut utiliser la syntaxe courte seulement si le nom de l'élément est «statique». On peut quand même avoir des attributs «dynamiques».

La syntaxe longue est obligatoire pour automatiser la construction d'éléments «dynamiques».

La syntaxe courte est plus concise, mais elle peut être plus difficile à relire si les expressions XPath sont complexes.

Exemple

Document XML

```
<?xml version="1.0" ?>
<Guide>
<Restaurant Categorie="**">
<Nom>Le Romantique</Nom>
<Adresse>
<Ville>Cabourg</Ville>
<Dept>Calvados</Dept>
                    </Adresse>
</Restaurant>
<Restaurant Categorie="***">
<Nom>Les TroisGros</Nom>
<Adresse>
<Ville>Roanne</Ville>
<Dept>Loire</Dept>
</Adresse>
</Restaurant>
</Guide>
```

Feuille de style XSLT

```
<?xml version="1.0" ?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/xsl">
 <xsl:template match="/">
<html><head><B>ESSAI XSL</B></head>
        <body><xsl:apply-templates/></body></html>
 </xsl:template>
 <xsl:template match="Guide">
        <H1>BONJOUR LE GROUPE XML</H1>
        <H2>SUIVEZ LE GUIDE</H2>
<xsl:apply-templates />
</xsl:template>
<xsl:template match="Restaurant">
<P> <I>Restaurant :</I>
<xsl:value-of select="Nom"/></P>
</xsl:template>
</xsl:stylesheet>
```

- Présentation générale
- Règles élémentaires
- Construction du document résultat
- **Structures itératives et conditionnelles**
- Variables XSL
- Règles paramétrées, et avec noms
- Autres aspects
- Conclusion

Balise xsl:apply-templates

On a besoin, à l'intérieur d'une règle, de faire appel à d'autres règles.

Cette balise permet de déclencher l'application de règles sur un ensemble de noeuds (les règles appliquées sont choisies selon le principe décrit précédemment).

- `<xsl:apply-templates/>`
  - La liste des nœuds à traiter est constituée des nœuds fils du nœud contexte.

- `<xsl:apply-templates select="path"/>`
  - La liste des nœuds à traiter est constituée des nœuds atteints par le chemin *path* depuis le nœud contexte.

Ajouter un mode à une template

- On peut associer un mode à une instruction `xsl:template` et activer une règle en utilisant un nom de mode spécifique dans `xsl:apply-templates` :

```
<xsl:template match="path" mode="name">
<xsl:apply-templates select="…"
mode="name">
```

Balise xsl:for-each

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

  applique les instructions à tous les noeuds désignés par le chemin XPath.

Exemple :

```
<xsl:template select=”bibliotheque”>
  <ul>
  <xsl:for-each select=”livre”>
    <li> Titre : <xsl:value-of select="titre" />.</li>
   </xsl:for-each>
   </ul>
</xsl:template>
```

Tri des résultats

- Par défaut on suit l'ordre du document
- On peut trier les noeuds d'un node-set en spécifiant un ou plusieurs axes de tri dans les instructions `xsl:apply-templates` et `xsl:for-each`

  `<xsl:sort select="`*`expression`*`" … />`
- Attributs possibles:
  - order : `ascending | descending`
  - Lang : "..."
  - data-type : `text | number`
  - case-order : `upper-first | lower-first`

Expressions conditionnelles (1)

- On peut tester la valeur (booléenne) d'une expression XPath et exécuter un template si la valeur est vraie : (en pratique on teste souvent l'existence d'un noeud)

```
<xsl:if test="expression">… </xsl:if>
```

Exemple classique :

```
<xsl:if test="livre">
  <ul>
  <xsl:for-each select="livre">
  <li>
    <xsl:value-of select="titre" />.
    <xsl:if test="@langue='français'">Ce livre est en français.</xsl:if>
  </li>
  </xsl:for-each>
 </ul>
```

Expressions conditionnelles (2)

Equivalent du switch en C : (sur un exemple)

```
<ul>
  <xsl:for-each select="livre">
    <li>
      <b><xsl:value-of select="auteur" /><br /></b>
      <xsl:value-of select="titre" />
      <xsl:choose>
        <xsl:when test="@langue='francais'">Ce livre est en français.</xsl:when>
        <xsl:when test="@langue='anglais'">Ce livre est en anglais.</xsl:when>
        <xsl:otherwise>Ce livre est dans une langue non répertoriée.</xsl:otherwise>
      </xsl:choose>
    </li>
  </xsl:for-each>
</ul>
```

- Présentation générale
- Règles élémentaires
- Construction du document résultat
- Structures itératives et conditionnelles
- **Variables XSL**
- Règles paramétrées, et avec noms
- Autres aspects
- Conclusion

Variables

- Une variable est un nom associé à une valeur (ce sont des variables non-mutables).

- La valeur peut être obtenue par une expression XPath ou bien par le contenu de l'élément `xsl:variable`.

- Pour obtenir la valeur d'une variable *$nom_var*
- Exemples :

```
<xsl:variable name='v1' select='12'/>
<xsl:variable name='v2' select='$v1+1'/>
<xsl:variable name='v3' select='@day'/>
<xsl:variable name='v4'>3</xsl:variable>
```

Variables (2)

- Une variable a une portée : l'élément dans lequel elle est déclarée et ses descendants.
- Exemple : `<date annee="2009" mois="octobre" jour="$v3"/>`
- On peut aussi enregistrer dans une variable un fragment du document d'entrée.
- Exemple : `<xsl:variable name="exposes" select="gdt/expose"/>`

- Présentation générale
- Règles élémentaires
- Construction du document résultat
- Structures itératives et conditionnelles
- Variables XSL
- **Règles paramétrées, et avec noms**
- Autres aspects
- Conclusion

Templates comme fonctions

- On peut aussi donner un nom à une règle et l'appeler comme un fonction:

```
<xsl:template name="nom"> ... </xsl:template>
```

```
<xsl:call-template name="nom"/>
```

- C'est une méthode qui permet d'utiliser plusieurs fois les mêmes éléments dans un document.

Paramètres (1)

- Un template peut avoir des paramètres (encore une fois comme une fonction).
- On déclare au début de la fonction les paramètres qu'elle utilise :
  <xsl:param name='p1'/>
- S'utilise en combinaison avec l'instruction <xsl:with-param>

Paramètres (2)

- Un exemple : définition de la fonction à un paramètre

```
<xsl:template name="polynome">
  <xsl:param name="variable_x" />
  <xsl:value-of
        select="2*$variable_x*$variable_x+(-5)*$variable_x+2" />
</xsl:template>
```

- Puis appel à la fonction :

```
<xsl:call-template name="polynome">
    <xsl:with-param name="variable_x" select="3.4" />
</xsl:call-template>
```

Paramètres (3)

- Un paramètre est très similaire à une variable. La seule différence est que sa valeur peut être modifiée.

- Un paramètre a un nom, une portée et une valeur par défaut. Pour obtenir sa valeur `$param`

```
<xsl:param name="p1" select="expr"/>
```

- Présentation générale
- Règles élémentaires
- Construction du document résultat
- Structures itératives et conditionnelles
- Variables XSL
- Règles «fonctions», et paramétrées
- **Autres aspects**
- Conclusion

Autres instructions

- Sur des ensembles de noeuds :
  - `<xsl:number select="...">` numérotation,
  - `<xsl:processing-instruction name="...">` construit un noeud instruction de traitement.

- `attribute-set`: pour grouper des déclarations d'attributs

- fonctions XPath additionnelles (dans le cas de documents multiples): `document, format-number, current`, ...

Exemple

```
<report><title>2004</title>
  <entry filename="wg-jan.xml"/>
  <entry filename="wg-feb.xml"/>
  <entry filename="wg-mar.xml"/>
</report>
```

```
<xsl:template match="/">
  <xsl:for-each select="/report/entry">
  <xsl:apply-templates
       select="document(@filename)"/>
  </xsl:for-each>
  </xsl:template>
```

Fragment résultat littéraux

- Un fragment résultat littéraux est un élément permettant d'émettre une chaîne de caractères (une constante) dans le flot de sortie. On peut émettre du texte, sans caractères de contrôle, mais aussi des éléments n'appartenant pas au namespace XSL

```
<xsl:text> ... </xsl:text>
```

- Ou directement le texte à l'intérieur du template
  (il doit être au moins dans une balise)

```
<xsl:template match="…"> <h1>du texte …
```

- On peut également insérer des commentaires

```
<xsl:comment> ... </xsl:comment >
```

Gestion des espaces inutiles

- On peut indiquer les éléments contenant des espaces inutiles en ajoutant l'instruction suivante:

```
<xsl:strip-space elements="path"/>
```

- Cette instruction indique à XSL « d'omettre » tous les nœuds textes se trouvant en-dessous de *path* et qui contiennent uniquement des espaces. Le comportement pour les nœuds contenant des caractères autre que des espaces n'est pas modifié
- On peut également se débarrasser des espaces se trouvant « aux extrémités » des nœuds textes:

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

Utilisation dans un butineur

- Les documents XML qui doivent être « directement transformés » au niveau du butineur peuvent contenir une instruction de traitement (P.I.) `xml-stylesheet` indiquant où trouver la feuille de style nécessaire
- Si la feuille de style est un document XSLT la valeur du pseudo-attribut `type` doit être `application/xml`

```
<?xml-stylesheet type="application/xml"
      href="http://www.lif.fr/file.xsl"?>
```

- **Remarque**: Microsoft Internet Explorer utilise `type="text/xsl"` bien que `text/xsl` ne soit pas un type de média MIME valide.

Exemple

```
<?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="facture">
      <xsl:apply-templates select="montant" />
   </xsl:template>
   <xsl:template match="montant">
   Le montant numéro <xsl:number count="montant" />
   vaut <xsl:value-of select="." />.
   </xsl:template>
 </xsl:stylesheet>
```

Exemple 2

```
<?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="CDlist">
<html><head><title> Exercice 1 </title></head><body>
Les compositions sont :
<ul><xsl:apply-templates select="CD/performance/composition"/>
</ul></body></html>
</xsl:template>
<xsl:template match="composition">
<li><xsl:value-of select="."/></li>
</xsl:template> </xsl:stylesheet>
```

- Présentation générale
- Règles élémentaires
- Construction du document résultat
- Structures itératives et conditionnelles
- Variables XSL
- Règles «fonctions», et paramétrées
- Autres aspects
- **Conclusion**

Quelques instructions vues

- xsl:template : définition d'une règle
- xsl:value-of, xsl:copy-of : insérer (valeur d') un noeud
- xsl:element, xsl:attribute : création de noeuds
- xsl:apply-templates, xsl:call-template : manipulations de règles
- xsl:for-each : boucles
- xsl:if, xsl:when, xsl:choose : tests
- xsl:text, xsl:comment, xsl:processing-instruction : insertion de noeuds spéciaux
- xsl:variable, xsl:parameter

TrAX

- Il n'existe pas d'API standard pour manipuler des feuilles de style XSLT qui soit utilisable sur plusieurs plateformes ou avec différents langages
- Il existe des API propriétaires comme par exemple TrAX (Transformations API for XML), inclus dans JAXP
  - existe uniquement pour Java
  - n'est pas supporté par tous les engins XSLT

XSL 2.0

- Recomandation W3C du 23 janvier 2007
- Utilisation de XPath 2.0, ce qui implique l'utilisation de séquences et des Schémas XML
- Possibilité d'avoir de multiples document résultats

```
<xsl:result-document href="file">
    template
</xsl:result-document>
```

- Output method pour XHTML
- Validation pour les schémas

Conclusion XSLT

- XML = format de présentation de données facilement échangeables sur le web
- production d'une source unique en XML
- XSLT permet d'adapter la présentation des données à n'importe quelle application

- XSLT peut être vu comme le "langage de programmation" de XML