4.*Transformation de documents XML avec* XSLT

• • •

XSL (eXtensible Stylesheet Language)

- **XSL** (*eXtensible Stylesheet Language*) est une famille de spécifications comprenant:
  - **XSLT** (pour *XSL Transformations*, langage de transformations)
  - **XSL-FO** (*XSL Formating Objects*, langage de présentation).

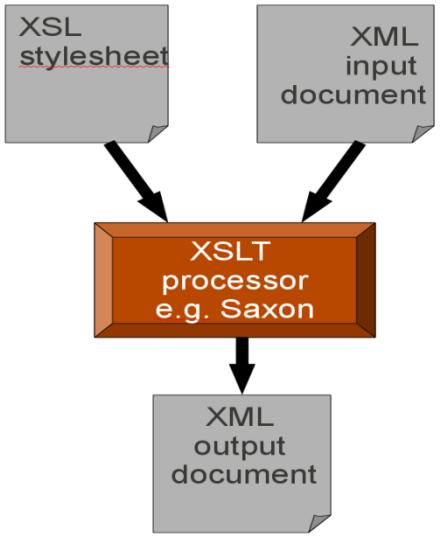
- XSLT est un langage permettant de produire un document XML ou texte à partir d'un autre document en appliquant des règles de transformation.

- *XSL*-FO (Extensible Stylesheet Language Formatting Objects) est une DTD XML qui définit la présentation d'un texte sur un document papier (PS, GV, PDF, DPS).

XSLT (extensible Stylesheet Langage Tansformations)

- Le langage XSLT est un dialecte XML permettant la transformation de documents

→ Extraire un contenu d'un document
pour le mettre dans un format reconnu
par une application de publication particulière



- Permet la génération d'autres contenus à partir d'un fichier XML, par exemple:
  - du HTML (bien formé)
  - du XML plus compliqué (tables de matière + règles de formattage XSL/FO)
  - des extraits en XML ou HTML ....
  - du SVG, X3D ou toutes sortes d'autres formats à partir de XML

Transformation typique

- A partir de ceci:

```
<source>

<title>XSL</title>
<author>John Smith</author>

</source>
```

- On veut produire:

```
<h1>XSL</h1>
<h2>John Smith</h2>
```

Comment? En XSL

- Une feuille de style XSL est un document XML

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

<xsl:template match="/">
   <h1>
      <xsl:value-of select="//title"/>
   </h1>
   <h2>
      <xsl:value-of select="//author"/>
   </h2>
</xsl:template>


</xsl:stylesheet>
```

Transformer un document

- XSLT permet de construire un nouveau document (en XML, HTML, etc) à partir d'un document XML existant en le transformant

- Extraire des fragments d'un document et les assembler différemment dans une structure nouvelle.

  - **À l'aide des feuilles de style**: un document XML qui contient un ensemble de **règles (template)**

  - **Chaque règle** décrit une transformation à appliquer à certains composants

- XSLT opère sur l'arbre (ordonné) du document source.

Feuille de style

- Chaque feuille de style XSL doit commencer par l'élément racine xsl:stylesheet.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <!--Mettre des règles de transformation-->

</xsl:stylesheet>
```

  - Les fichiers XSLT ont l'extension *.xsl
  - L'attribut version='1.0' **(obligatoire)** précise la version de la spécification XSL(T).
  - Attribut xmlns:xsl : espace de nom XSL
  - La feuille de style est contenue dans l'élément racine xsl:stylesheet.

Feuille de style

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl ="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
    <body>
    Diverses balises Html et XSL... Par exemple :
    <xsl:value-of select="chemin d'accès/élément" />
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Feuille de style

- Élément <xsl:output>: Format de sortie du document résultat

```
<xsl:output method="xml" version="1.0"
     encoding="UTF-8" indent="yes"/>
```

  - Attribut method: type de document en sortie
  - Attribut encoding: codage du document
  - Attribut indent: indentation en sortie

- Différents types de document en sortie:
  - Xml: vérifie que la sortie est bien formée (sortie par défaut)
  - Html: accepte les balises manquantes, génère les entité HTML. (Sortie par défaut si XSL reconnait l'arbre de sortie HTML4)
  - Text: tout autre format textuel: du code Java, Latex, etc.

Règles

- Une feuille de style contient une suite de **règles** (*xsl:template*)

- L'application de la feuille de style consiste à:
  - Parcourir le document source en partant de la racine
  - Pour chaque nœud rencontré dans ce parcours, chercher la règle à appliquer
  - L'application d'une règle produit un fragment du document résultat

Modes d'utilisation

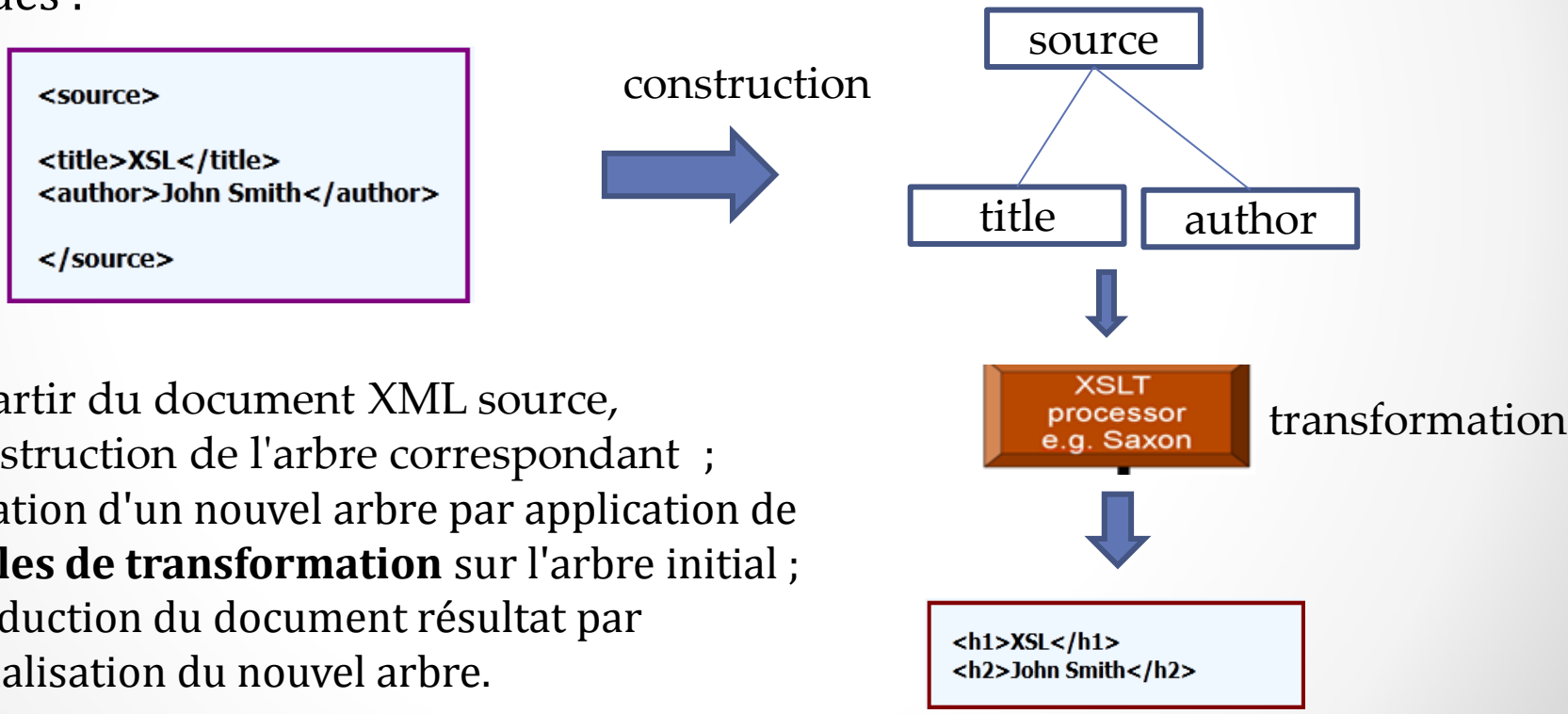
- Navigateurs (coté client)
  - Doivent être dotés d'un processeur XSLT

- Processeurs en ligne de commande (et/ou API)
  - programmes appelés manuellement. Ces programmes prennent comme paramètres le fichier XML source, le fichier XSLT et produisent le fichier résultant de la transformation.
  - Xalan du projet Apache
  - Saxon
  - msxsl (un processeur XSLT pour Windows)

Processeur XSLT

- **Etapes du processus**

Lorsqu'un processeur XSLT est invoqué, plusieurs traitements sont effectués :



- à partir du document XML source, construction de l'arbre correspondant ;
- création d'un nouvel arbre par application de **règles de transformation** sur l'arbre initial ;
- production du document résultat par sérialisation du nouvel arbre.

Exemple

```xml
<?xml version="1.0"?>
<personnes>
    <personne>
        <nom> Martin </nom>
        <prenom> Jacques </prenom>
        <adresse Nrue="25"> Paris </adresse>
    </personne>
</personnes>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
    <!-- Règle1 -->
    <xsl:template match="/">
        <html><body>
            <xsl:apply-templates/>
        </body></html>
    </xsl:template>

    <!-- Règle2 -->
    <xsl:template match="personnes">
        <h1>PERSONNALITES</h1>
        <xsl:apply-templates/>
    </xsl:template>

    <!-- Règle3 -->
    <xsl:template match="personne">
    <h2>Nom:  <xsl:value-of select="nom"/></h2>
    <h2>Prenom:  <xsl:value-of select="prenom"/></h2>
    Adresse : <p> <xsl:value-of select="adresse/@Nrue"/> <xsl:value-of select="adresse"/>  </p>
    </xsl:template>
</xsl:stylesheet>
```

Règle1

```xml
<html>
<body>
    <h1>PERSONNALITES</h1>
    <h2>Nom:    Martin </h2>
    <h2>Prenom:    Jacques </h2>
Adresse :
    <p>25 Paris </p>
</body>
</html>
```

Règle2

Règle3

Définition d'une Règle XSLT

- **<xsl:template>** (enfant de <xsl:stylesheet>): permet de définir une règle et précise par un motif XPath, les nœuds sur lesquels elle s'applique.

Syntaxe:

```
<xsl:template
      match = Pattern
      priority = number
      mode = QName>
 <!-- Modèle de transformation-->

</xsl:template>
```

- **Le Pattern** permet d'atteindre des nœuds cibles de la transformation. S'exprime sous forme d'une expression XPath (*l'attribut match*).

- **Le modèle de transformation** décrit ce par quoi il faut remplacer le sous-arbre que le pattern désigne (ou les sous-arbres si le motif en désigne plusieurs).

Exemple simple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html>
      <head>
        <title> Ma première règle </title>
      </head>
      <body>
        <h1>Bonjour tout le monde !</h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Motif: filtre la racine du document d'entrée

Modèle « document html »

```
<html>
    <head>
        <title> Ma première règle </title>
    </head>
    <body>
        <h1>Bonjour tout le monde !</h1>
    </body>
</html>
```

Les patterns (motifs)

- **Un pattern** est une expression qui, évaluée par rapport à un certain noeud contexte, désigne un certain ensemble de noeuds de l'arbre XML d'un document.

- On ne peut pas associer n'importe quelle expression XPath à l'attribut match
  - Certaines expressions seraient trop complexes à évaluer
  - L'expression doit toujours désigner un ensemble de nœuds

```
<xsl:template match="1">
```
NON

```
<xsl:template match="preceding::node()[5]">
```
NON

- Les axes possibles:
  - Child→ les nœuds enfants d'un élément
  - Attribute → les attributs d'un élément
  - // → les descendants d'un élément ou lui-même (descendant-or-self::node())
  - Dans un prédicat, aucun type d'axe de localisation n'est interdit

Modèle de transformation

- Décrit ce par quoi il faut remplacer le sous-arbre que le motif désigne (ou les sous-arbres si le motif en désigne plusieurs).

  →Construction de l'arbre résultat

- Comprend du texte et différentes *instructions XSLT*
  - *Instruction fondamentale* xsl:apply-templates
  - *Instructions de constructions de noeuds*

Instruction xsl:apply-templates

- Permet d'**appliquer** explicitement une règle sur une séquence de nœuds.
- Syntaxe:

```
<xsl:apply-templates
    select = Expression
    mode = QName >
</xsl:apply-templates>
```

- *Sans attributs*: les règles seront appliquées **à tous les fils du nœud contexte**
- Attributs *select*: contenant une expression Xpath: règles appliquées sur les nœuds sélectionnés par l'attribut select
- Attribut *mode*
  - Permet de choisir explicitement une des règles parmi celles qui sont candidates

→Un nœud peut être traité plusieurs fois pour générer un résultat différent à chaque fois

→Produire plusieurs résultats à partir d'un noeud

```
<xsl:apply-templates mode="nom-de-mode" />
```

Exemple

```
<exemple>
    <titre>ceci est un titre</titre>
    <auteur>ceci est un auteur</auteur>
</exemple>
```

```
<html>
    <head>
    <title> transformer le document exemple </titre>
    </head>
    <body>
      <h1>ceci est un titre</h1>
      <h1>ceci est un auteur</h1>
    </body>
</html>
```

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
    <xsl:template match="/">
        <html>
        <head> <title> transformer le document exemple</title> </head>
        <body> <xsl:apply-templates/> </body>
    </html>
    </xsl:template>

    <xsl:template match="titre">
        <h1> <xsl:value-of select="."/> </h1>
    </xsl:template>

    <xsl:template match="auteur">
        <h1> <xsl:value-of select="."/> </h1>
    </xsl:template>

</xsl:stylesheet>
```

Règle 1: s'applique à la racine '/' crée la structure du document XHTML.

Examine tous les noeuds enfants dans l'ordre

Règle 2: s'applique à l'élément titre

Règle 3: s'applique à l'élément auteur

Exemple

```
<exemple>
    <titre>ceci est un titre</titre>
    <auteur>
        <nom>nom auteur </nom>
        <prenom>prenom auteur </prenom>
    </auteur>
</exemple>
```

```
<html>
  <head>
    <title> transformer le document exemple</title>
  </head>
  <body>
    <h1>ceci est un titre</h1>
    <h1>nom auteur </h1>
  </body>
</html>
```

```
<xsl:template match="/">
    <html>
        <head> <title> transformer le document exemple</title> </head>
        <body> <xsl:apply-templates select="//titre"/>
        <xsl:apply-templates select="//nom"/>  </body>
    </html>
</xsl:template>

<xsl:template match="titre">
    <h1> <xsl:value-of select="."/> </h1>
</xsl:template>

<xsl:template match="nom">
    <h1> <xsl:value-of select="."/> </h1>
</xsl:template>
```

Construction de contenu

- Chaque application de règle de la feuille de style produit un fragment du résultat.

- Ce fragment est construit à partir du contenu de l'élément xsl:template et d'autres éléments permettant d'insérer d'autres nœuds calculés.

- Construction de contenu
  - Noeud textuel par Xpath
  - Texte brut
  - Nœuds élément et attribut
  - Liste d'attributs
  - Copie de nœuds
  - Commentaire et instruction de traitement
  - etc

Nœud textuel par XPath

- Une règle XSLT utilisant l'instruction *<xsl:value-of>*
- Syntaxe:

```
<xsl:value-of
    select = Expression
    disable-output-escaping = "yes" | "no" >
</xsl:value-of>
```

- *<xsl:value-of select="..." />* est remplacée lors de l'instanciation du modèle par la valeur textuelle de ce qui est désigné par l'attribut select (obligatoire).

  → Extraction du contenu de l'arbre en entrée

- disable-output-escaping (optionnel): pour le traitement des caractères spéciaux. Par exemple ">" sera affiché ">" si cette propriété est à "yes"

Exemple

Saison.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Saison>
    <Concert>
        <Organisation> Anacréon </Organisation>
        <Date>Samedi 9 octobre 1999 <Heure> 20H30 </Heure> </Date>
        <Lieu>Chapelle des Ursules</Lieu>
    </Concert>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mardi 19 novembre 1999 <Heure> 21H </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
    <Théâtre>
        <Organisation> Masques et Lyres </Organisation>
        <Date>Mercredi 20 novembre 1999 <Heure> 21H30 </Heure> </Date>
        <Lieu>Salle des Cordeliers</Lieu>
    </Théâtre>
</Saison>
```

Saison.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version = "1.0">

    <xsl:output method='text' encoding='UTF-8'/>

    <xsl:template match='/'>
        Date Concert : <xsl:value-of select="Saison/Concert/Date"/>
        Date Théâtre : <xsl:value-of select="Saison/Théâtre[1]/Date"/>
        Date Théâtre : <xsl:value-of select="Saison/Théâtre[2]/Date"/>
    </xsl:template>

</xsl:stylesheet>
```

Résultat

```
Date Concert : Samedi 9 octobre 1999  20H30
Date Théâtre : Mardi 19 novembre 1999  21H
Date Théâtre : Mercredi 20 novembre 1999  21H30
```

Instruction xsl:value-of et type noeud

- Le nœud sélectionné est un *élément*
  - Concaténation de tous les textes qui se trouvent comme contenu de cet élément et de ses descendants.
- Le nœud est un nœud *text*
  - Texte du nœud lui-même
- Le nœud est un *Attribut*
  - Valeur de l'attribut
- Le nœud est une *instruction de traitement*
  - Valeur de l'instruction de traitement
- Le nœud est un *commentaire*
  - Le texte du commentaire (sans les marques <!– et-->)

Exemple

- En entrée:

```
<carteDeVisite>
<nom>Martin</nom>
</carteDeVisite>
```

- Règle:

```
<xsl:template match="carteDeVisite">
<p>Nom : <xsl:value-of select="nom"/></p>
</xsl:template>
```

- En sortie:

Exemple

- En entrée:

```
<note>enseigne <clé>XML</clé> au Master</note>
```

- Règle:

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

- En sortie:

Exemple

- En entrée:

```
<note>enseigne <clé>XML</clé> au Master</note>
```

- Règle:

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

- En sortie:

Texte brut

- L'élément *<xsl:text>* utilise son contenu pour créer un nœud textuel dans le document résultat.

- Syntaxe:

```
<xsl:text disable-output-escaping = "yes" | "no">
  ....Texte....
</xsl:text>
```

- Exemple:

```
<root>
    <!-- document quelconque -->
</root>
```

```
<xsl:template match="/">
    <xsl:text>
    un texte
       avec des espaces &amp;amp; des sauts de ligne
    </xsl:text>
    <xsl:text  disable-output-escaping="yes">
       un autre texte avec des espaces &amp;amp; des sauts de ligne
    </xsl:text>
</xsl:template>
```

```
un texte
avec des espaces &amp;

des sauts de ligne

un autre texte
avec des espaces &

des sauts de ligne
```

Exercice

- family.xml

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="family.xsl"?>
<family>
  <person>
    <given-name age="10">Fred</given-name>
    <family-name>Smith</family-name>
  </person>
  <person>
    <given-name age="13">Jill</given-name>
    <family-name>Jones</family-name>
  </person>
</family>
```

- family. xsl

En sortie

```xml
<?xml version="1.0" encoding="UTF-16"?>
  <p>Fred Smith</p>
  <p>Jill Jones</p>
```

Nœud élément

- Quand on ne connait pas le nom de l'élément à insérer
- <xsl:element> Crée un élément de sortie et lui donne le nom spécifié (nom calculé dynamiquement).

  Syntaxe:

```
<xsl:element
        name = "element-name"
        namespace = "uri-reference"
        use-attribute-sets = QName>
</xsl:element>
```

  - *Name (attribut obligatoire):* nom de l'élément à créer
  - *Namespace (optionnel):* URI d'espace de noms de l'élément créé.
  - *Use-attribute-sets:* Liste d'ensembles d'attributs, séparés par des espaces

Exemple

```
<xsl:template match="part">
    <xsl:element name="partie">
        <xsl:value-of select="title"/>
    </xsl:element>
</xsl:template>
```

```
<?xml version="1.0"?>
<part>
  <title>Le titre</title>
</part>
```

```
<xsl:template match="part">
    <partie>
        <xsl:value-of select="title"/>
    </partie>
</xsl:template>
```

Exemple

```
<personne>
<nom>Martin </nom>
<prenom>Jacques</prenom>
</personne>
```

```
<xsl:template match = "personne" >
    <xsl:element name = "identite" >
        <xsl:element name = "{nom}"/>
        <xsl:element name = "{prenom}"/>
    </xsl:element>
</xsl:template>
```

```
<identite>
    <Martin/>
    <Jacques/>
</identite>
```

Résultat

Nœud attribut

- <xsl:attribute> Crée un nœud d'attribut et le joint à un élément de sortie.

  Syntaxe:

```
<xsl:attribute
  name = "attribute-name"
  namespace = "uri-reference">
</xsl:attribute>
```

  - *Name (attribut obligatoire):* nom de l'attribut à créer
  - *Namespace (optionnel):* URI d'espace de noms de l'attribut créé.

Exemple

```
<personne>
<nom>Martin </nom>
<prenom>Jacques</prenom>
</personne>
```

```
<xsl:template match = "personne" >
    <xsl:element name = "presentateur" >
        <xsl:attribute name = "{name()}">
            <xsl:value-of select="nom"/>
            <xsl:text>  </xsl:text>
            <xsl:value-of select="prenom"/>
        </xsl:attribute>
    </xsl:element>
</xsl:template>
```

Liste d'attributs

- *<xsl:attribute-set>:* Définit un ensemble nommé d'attributs.

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

```
<xsl:attribute-set
  name = QName
  use-attribute-sets = QNames>
</xsl:attribute-set>
```

  - *Name (attribut obligatoire):* nom de l'ensemble d'attributs
  - *Use-attribute-sets:* Liste d'ensembles d'attributs, séparés par des espaces

- Les listes sont définies en dehors des règles.

Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:attribute-set name="MonStyle">
        <xsl:attribute name="bgcolor">white</xsl:attribute>
        <xsl:attribute name="font-name">Helvetica</xsl:attribute>
        <xsl:attribute name="font-size">18pt</xsl:attribute>
    </xsl:attribute-set>
    <xsl:template match="/">
        <html>
        <head><title>Ficher personne</title></head>
        <body xsl:use-attribute-sets="MonStyle">
        Exemple de use-attribut-sets
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Résultat

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Ficher personne</title>
</head>
<body bgcolor="white" font-name="Helvetica" font-size="18pt">
   Exemple de use-attribut-sets
</body>
</html>
```

Copie de nœud

- **Copie du nœud courant (sans attributs)** dans le document résultat:

```
<xsl:copy
  use-attribute-sets = QNames>
</xsl:copy>
```

- **Copie de nœuds:**

```
<xsl:copy-of select = Expression />
```

est instanciée comme une copie conforme des éléments sélectionnés

  - permet de copier des nœuds sélectionnés ainsi que tous les descendants de ces nœuds (nœuds d'attributs, espaces de noms et les enfants du nœud d'élément) dans le document résultat.

Exemple

- family.xml

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="family.xsl"?>
<family>
  <person>
    <given-name age="10">
      <name>Fred</name>
      <nick-name>Freddy</nick-name>
    </given-name>
    <family-name>Smith</family-name>
  </person>
  <person>
    <given-name age="10">
      <name>Robert</name>
      <nick-name>Bob</nick-name>
    </given-name>
    <family-name>Smith</family-name>
  </person>
</family>
```

- family.xsl

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

<xsl:template match="person">
  <p>
    <xsl:copy-of select="given-name"/>
    <xsl:text> </xsl:text>
    <xsl:copy-of select="family-name"/>
  </p>
</xsl:template>

</xsl:stylesheet>
```

Commentaire

- *<xsl:comment>* Génère un commentaire dans la sortie.
- Syntaxe:

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

- Exemple

```
<?xml version ="1.0"?>
<?xml-stylesheet type="text/xsl" href="news.xsl"?>
<news>
<story1>Here is the top news story.</story1>
    <story2> Here is the next news story.</story2>
</news>
```

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

<xsl:template match="/">
<HTML>
<BODY>
<xsl:comment>insert top news story</xsl:comment>
<P>
<xsl:value-of select="//story1"/>
</P>
</BODY>
</HTML>
</xsl:template>

</xsl:stylesheet>
```

En sortie

```
<HTML>
<BODY><!--insert top news story>
<P>Here is the top news story.</P>
</BODY>
</HTML>
```

Priorité entre règles

- Quelle règle choisir dans le cas de plusieurs règles éligibles?
- La priorité peut être spécifiée explicitement avec l'attribut **priority.**
- Sinon c'est la règle la plus spécifique qui est choisie

```
<xsl:template match="Personne">
  <regle><xsl:apply-templates/></regle>
</xsl:template>

<xsl:template match="Personne[Bureau]">
   <Autreregle><xsl:apply-templates/></Autreregle>
</xsl:template>
```

```
<?xml version="1.0"?> …
<table><description>personnel 4ème étage</description>
   <personne><nom>Bond</nom><bureau>U1</bureau></personne>
   <personne><nom>Lupin</nom></personne>
   <personne><nom>Templar</nom><bureau>U3</bureau></personne>
</table>…
```

Pour `<personne>…<personne>` → la première règle s'applique

Pour `<personne>…<bureau>…</bureau><personne>` → Seule la seconde s'applique

XSLT: fonctions étendues

- Les instructions de transformation:
  - Parcours itératifs: *xsl:for each*
  - Tri: *xsl:sort*

- Les instructions de programmation:
  - Traitement conditionnel: *xsl:if, xsl:choose, xsl:when*
  - Variables: *xsl:variable*
  - Templates nommés

Instruction xsl:for-each

Structure de répétition xsl:for-each

- parcourir un ensemble de noeuds sélectionnés avec select
- Les instructions sont appliquées successivement à chaque noeud sélectionné

Pas de variable, donc pas d'incrémentation

Syntaxe:

```
<xsl:for-each
  select = Expression>
</xsl:for-each>
```

Exemple

```xml
<?xml version="1.0" encoding="utf-8"?>
<Texte>
    <Chapitre>
        <Titre>titre1</Titre>
        <Section>
            <Titre>titresect1</Titre>
        </Section>
    </Chapitre>
    <Chapitre>
        <Titre>titre2 </Titre>
        <Section>
            <Titre>titresect2</Titre>
        </Section>
    </Chapitre>
</Texte>
```

```xml
<xsl:template match="/">
    <R>
        <xsl:for-each select=".//Titre">
            <H1>
             <xsl:apply-templates/>
            </H1>
        </xsl:for-each>
    </R>
</xsl:template>
```

Tri: *xsl:sort*

- Instruction de tri
- Permet de trier de nœuds sélectionnés par les instructions *xsl:apply-templates* ou *xsl:for-each*
- À placer après la balise ouvrante de *xsl:for-each* ou *xsl:apply-templates.*
- Par défaut, l'ordre du tri est croissant (éléments ordonnés suivant l'ordre lexicographique de la valeur textuelle de chaque élément)
- En l'absence d'une instruction *<xsl:sort/>*, *<xsl:for-each>* et *<xsl:apply-templates>* constituent une liste des éléments à traiter, basée sur **l'ordre naturel de lecture** du document XML.

Critères de Tri: *xsl:sort*

- Syntaxe

```
<xsl:sort
  select = string-expression
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" }
  />
```

- Utilisation des attributs: *select, order, case-order*, etc.
  - **Select:** définit la clé du tri. Prendra comme valeur une expression XPath (valeur par défaut (.) qui est la valeur textuelle du nœud courant).
  - **Order:** définit l'ordre du tri (ascendant ou descendant). Peut prendre l'une des 2 valeurs *ascending* (valeur par défaut) ou *descending*.
  - **Case-ordre:** définir la relation d'ordre entre les lettres minuscules et majuscules. Peut prendre les valeurs *upper-first* ou *lower-first*. (valeur par défaut dépend de la langue utilisée).

Exemple

```
<Texte>
    <Chapitre>
        <Titre>titre1</Titre>
        <Section>
            <Titre>titresect1</Titre>
        </Section>
    </Chapitre>
    <Chapitre>
        <Titre>titre2 </Titre>
        <Section>
            <Titre>titresect2</Titre>
        </Section>
    </Chapitre>
</Texte>
```

```
<xsl:template match="/">
 <R>
    <xsl:for-each select=".//Titre">
        <xsl:sort select="."/>
        <H1><xsl:apply-templates/></H1>
    </xsl:for-each>
 </R>
</xsl:template>
```

Traitement conditionnel : *xsl:if*

- Permet le traitement conditionnel d'un nœud
- Syntaxe: <xsl:if test="condition-booléenne"> Instructions... </xsl:if>

En entrée

```
<carnetDAdresse>
  <carteDeVisite>
    <nom>Bekkers</nom>
    ...
  </carteDeVisite >
  <carteDeVisite>
    <nom> Bartold </nom>
    ...
  </carteDeVisite >
  ...
</ carnetDAdresse >
```

```
<xsl:for-each select="carteDeVisite">
  <xsl:value-of select="nom"/>
  <xsl:if test="position()!=last()">,
  </xsl:if>
</xsl:for-each>
```

- Génère une virgule après chaque nom sauf pour le dernier
- Pas de else

Traitement conditionnel : *xsl:choose*

- Test à choix multiples (équivalent au switch/case du C)

- <xml:choose> avec : <xsl:when> et <xsl:otherwise>

```
<xsl:choose>
  <xsl:when test="quelque-chose">
    [action]
  </xsl:when>
  <xsl:when test="autre-chose">
    [action]
  </xsl:when>
    ...
  <xsl:otherwise>
    [action]
  </xsl:otherwise>
</xsl:choose>
```

Exemple

- Le fragment de feuille de style retourne le contenu de l'enfant title de nœud courant si cet enfant existe ou construit un titre avec un numéro sinon.

```
<xsl:choose>
  <xsl:when test="title">
    <xsl:value-of select="title"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:text>Section </xsl:text>
    <xsl:number level="single" count="section"/>
  </xsl:otherwise>
</xsl:choose>
```

Variables et paramètres

- Variables:
  - Les variables servent à stocker des valeurs (atomique, un nœud ou une suite de ces valeurs).
  - Elles peuvent être utilisées dans les expressions XPath.
  - Élément: < xsl:variable>

- Paramètres:
  - servent à transmettre des valeurs aux règles (l'élément xsl:param) .
  - L'élément xsl:with-param permet d'instancier un paramètre lors de l'appel à une règle.

- La principale différence entre une variable et un paramètre est qu'un paramètre peut être passé comme argument à un template.

Variables : *xsl: variable*

- XSLT permet de définir des variables pouvant stocker des valeurs.

```
<xsl:variable
    name = QName
    select = Expression>
</xsl:variable>
```

- Les variables sont visibles dans toute la sous arborescence

- Une variable, en XSLT comme dans tout autre langage, est l'association d'un nom et d'une valeur. Néanmoins, en XSLT, cette association est indestructible : il est impossible de changer la valeur d'une variable, une fois qu'on l'a déterminée.

- L'attribut name détermine le nom de la variable. La valeur est donnée soit par une expression XPath dans l'attribut select soit directement dans le contenu de l'élément xsl:variable

Exemple

```
<xsl:variable name="v1" select="12"/>
```

→ Attribut select avec la valeur (une constante)

```
<xsl:variable name="v1" select="/COURS/ENSEIGNANTS"/>
```

→Attribut select avec une expression XPath

→ Valeur = contenu du fils de ENSEIGNANTS dans l'arbre

Exemple

```
<personne>
<nom>
    <prenom>Jacques</prenom>
    <famille>Martin</famille>
</nom>
</personne>
```

```
<xsl:template match="/">
    <xsl:variable name="nom" select="personne/nom/prenom"/>
    <xsl:copy-of select="$nom"/>
</xsl:template>
```

Paramètres : *xsl: param*

- les paramètres représentent un type particulier de variables
- Servent à transmettre des valeurs à la feuille de style et aux règles
- L'élément param peut être enfant de l'élément racine xsl:stylesheet ou des éléments xsl:template
- L'exemple déclare un paramètre bg-color avec une valeur par défaut égale à la chaîne de caractères white

```
<xsl:param name="bg-color" select="'white'"/>
```

- Ou

```
<xsl:param name="bg-color">white<xsl:param/>
```

Passage de Paramètres : *xsl: with-param*

- Transmet un paramètre à un modèle
- Fils de <xsl:apply-templates> et <xsl:call-template>
- Syntaxe:

```
<xsl:with-param
  name = QName
  select = Expression>
</xsl:with-param>
```

  - **Name:** Obligatoire. Le Noms qualifiés du paramètre.
  - **Select:** Une Expression à comparer au contexte actuel. Il n'y a pas de valeur par défaut. En l'absence de contenu, une chaîne vide est générée.

Templates nommés et fonctions

- XSL permet de nommer un template et de l'appeler explicitement à n'importe quel endroit

→ Factorisation de code

- Déclaration de fonction:

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

- Appel de fonction

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

- On peut passer des paramètres avec xsl:param

Templates nommés et fonctions

```
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"></xsl:output>

<xsl:template match = "/" >
    <xsl:call-template name = "print" />
</xsl:template>
<xsl:template name = "print" >
    <XXX >xxx</XXX>
</xsl:template>

</xsl:stylesheet>
```

Déclaration de feuilles de style

- Importation de feuilles XSL

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

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

Syntaxe

```
<xsl:import href = "uri-reference"/>
```

- Inclusion de feuilles XSL
  - Syntaxe

```
<xsl:include href = "uri-reference"/>
```

  - **Href:** Obligatoire. Référence URI (Uniform Resource Identifier) identifiant le fichier XSLT à inclure.
  - <xsl:include> est enfant de l'élément <xsl:stylesheet>

Conclusion

- XSLT est un vrai langage de règles pour la transformation de documents
- Basé sur le langage XPath
- Une feuille de style pour *un document (ou une classe de documents)*

Liens utiles

- XSLT recommandation W3C:http://xmlfr.org/w3c/TR/xslt/

- XSLT version 2.0 *http://www.w3.org/TR/xslt20/*
- Les éléments XSLT: http://msdn.microsoft.com/fr-fr/library/ms256058%28v=vs.80%29.aspx

- Support de Cours en ligne Elisabeth Murisasco

- Jacques Le Maitre, Description et manipulation de documents XML, supports de cours en ligne http://lemaitre.univ-tln.fr/cours.htm