

Gestion des documents Structurés et Multimédia

XSLT

Stéphane Bonhomme

stephane@exselt.com

Langages de présentation

- **Feuilles de style CSS : Cascading Style Sheet**
 - Attachement de propriétés aux éléments
 - Pour les documents (X)HTML et XML
 - Le moyen le plus simple pour obtenir des présentations web adaptables et évolutives
 - Syntaxe non XML
- **XSL-FO : eXtensible Stylesheet Language-Formatting Objects**
- **SMIL : Synchronous Multimedia Integration Language**
- **SVG : Scalable Vector Graphics**

Ressources sur les langages de présentation

- **Site du w3c sur les styles:**
<http://www.w3.org/Style/>
- **Version française du standard xslt :**
<http://xmlfr.org/w3c/TR/xslt/>
- **Éditeurs & processeurs XSLT (xalan, saxon)**
<http://xml.coverpages.org/xslSoftware.html>
- **FOP (Formatting Objects Processor) : FO vers PDF** **<http://xml.apache.org/fop/>**
- **CSS-DOM & SAC (Simple API for CSS)**

Feuille de style CSS

- Insertion au début du source XML

BookCatalogue.xml

```
<?xml-stylesheet type="text/css" href="monStyle.css"?>
```

- **Exemple de règles css contenues dans monStyle :**

```
BookCatalogue { display: block; margin: 5%;  
                font-size: 12pt }
```

```
Book           { display: list-item; list-style: square  
                }
```

```
Title         { display: block; font-size: large }
```

```
Author        { display: block; text-indent: 2em }
```

```
ISBN          { display: block; text-indent: 2em;  
                font-weight: bold }
```

```
Date, Publisher{ display: none }
```

eXtensible Stylesheet Language

- **Contexte : approche déclarative des documents**
- **Définition de structures de base de formatage appelées « objets de flux » (caractère, paragraphe, séquence, page, groupe, lien, etc.).**
- **Mécanisme de transformation d'arbre : spécification de comment chaque élément de l'arbre source est associé aux objets de flux de l'arbre cible.**
- **Langage de style complet pour réordonner, dupliquer, changer la structure des éléments et construire des pages complexes (colonnes, tables des matières, etc.).**

Introduction à XSLT

- **XSLT est une recommandation du W3C (nov. 1999)**
- **Actuellement version 1, version 2 en préparation**
- **Facilite l'interopérabilité entre applications en facilitant la conversion des données XML**
- **Permet une présentation des données xml : en (x)html, ou en pdf (via xslfo)**
- **Les feuilles de transformation XSLT sont des documents xml, régis par un schéma.**

Feuilles de style XSLT

- **Un XSLT est un document XML**

- Bien formé
- Déclaration XML

`<?xml version="1.0"?>`

- Déclaration des espaces de nommage

`<xsl:stylesheet`

`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`

`xmlns:fo="http://www.w3.org/1999/XSL/Format">`

- Contient

- Instructions de conduite de la transformation

- Namespace xsl

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

- Balises et contenu à produire dans le doc cible

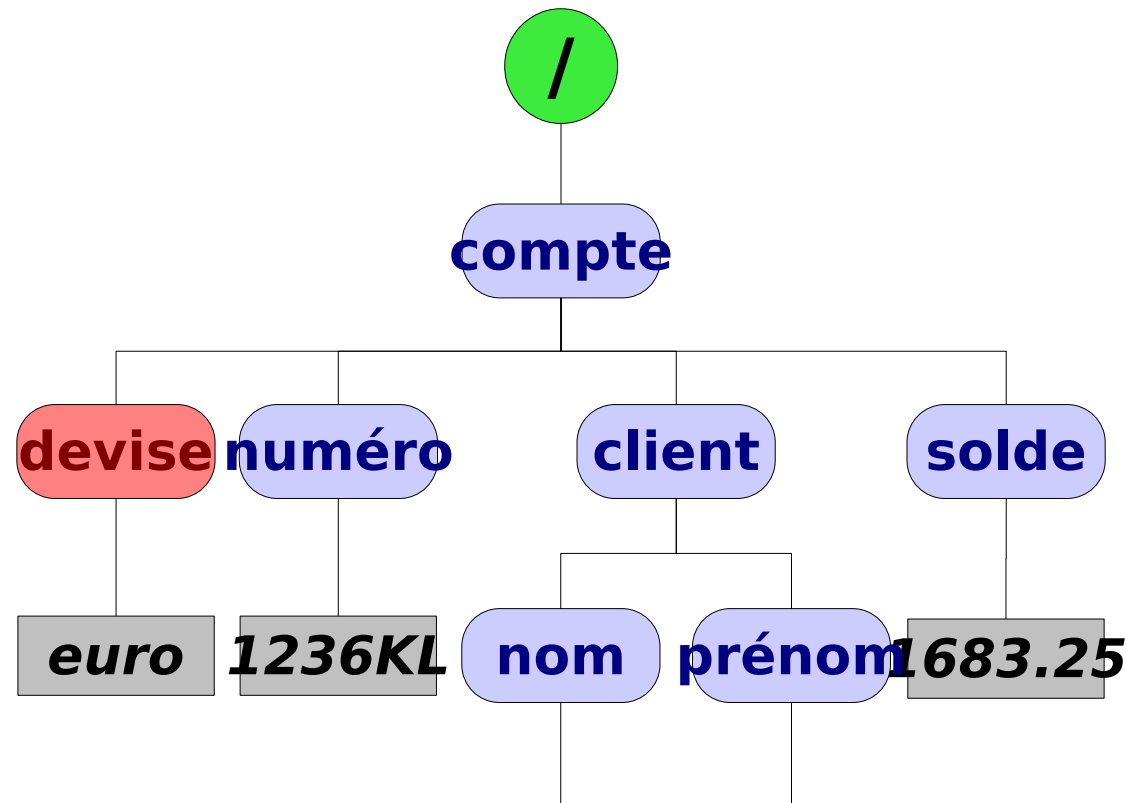
- Autres namespaces

`<fo:root>`

Représentation des données

- Représentées par un arbre

```
<compte devise="euro">  
  <numéro>123456KL</numéro>  
  <client>  
    <nom>Dupont</nom>  
    <prénom>Michel</prénom>  
  </client>  
  <solde>1683.25</solde>  
</compte>
```



Principes

- **Une feuille de style xslt est composée de templates**
 - Correspondant à un ou des noeuds de l'arbre source
 - Dont le contenu spécifie comment traiter ces noeuds

```
<xsl:stylesheet version='1.0'
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="compte">
    <!--règle de production pour compte-->
  </xsl:template>
  <xsl:template match="client">
    <!--règle de production pour client-->
  </xsl:template>
</xsl:stylesheet>
```

Déroulement de la transformation

- **Parcours de l'arbre XML source dans l'ordre préfixe**
- **A chaque noeud élément rencontré, recherche d'un template applicable :**
 - Pas de template applicable : poursuite du parcours
 - 1 template applicable : application de la règle et poursuite du parcours après la fin de la balise source
 - Plusieurs templates : application de template le plus spécifique
- **Après l'application d'un template**
 - Poursuite du parcours sur les noeuds suivants
 - Pas de parcours de la descendance d'un noeud traité
- **Les noeuds texte sont copiés dans le résultat**

Exemple

Document XML source

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl"
    href="bonjour.xsl"?>

1 <biblio>
2   <book lang="fr">
      <title>XML</title>
      <auteur>Jean Martin</auteur>
4   </book>
5   <book lang="en">
      <title>XML in action</title>
      <author>John Smith</author>
7   </book>
8   <paper>
9     <title>XML software</title>
11    <date>05/02/2002</date>
12  </paper>
</biblio>
```

Feuille de style XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns="...">
  <xsl:template match="book">
    <p>a book</p>
  </xsl:template>

  <xsl:template
    match="book[lang='en']">
    <p>a book in english </p>
  </xsl:template>

  <xsl:template match="title">
    <p>a title</p>
  </xsl:template>
10 </xsl:stylesheet>
```

Résultat de la

```
<p>a book</p>
<p>a book in english </p>
<p>a title</p>
05/02/2002
```

Règles de production

- **Spécifie la transformation d'un noeud de l'arbre source**
 - Ce noeud de vient le *noeud courant*
 - Contient :
 - Des instructions XSL (espace de nommage XSL)
 - Contrôle du processus de transformation
 - Génération de balises et d'attributs
 - Copie d'éléments source vers cible
 - Des balises à produire
 - Des feuilles de texte

```
<xsl:template match="book">  
  <p>Titre : <xsl:value-of select="title"/></p>  
  <xsl:apply-templates select="authors"/>  
</xsl:template>
```

Exercice

- **Écrire la feuille de style qui produit le texte intégral du fichier content.xml du document odp utilisé pour le TP1**
 - Tout noeud ne correspondant pas à un template est parcouru, sans produire de sortie
 - Toute feuille de texte est reproduite dans la sortie
- **Pour appliquer une feuille de style :**
 - En shell :
`xsltproc feuille.xsl document.xml > resultat.xml`
 - Dans le navigateur :
en ajoutant la référence à la feuille de style à l'en-tête du fichier xml
`<?xml-stylesheet type="text/xsl" href="feuille.xsl"?>`

Noeud Courant

- Lors de l'application d'un template, le noeud courant est le noeud de l'arbre source reconnu par l'attribut *match*
- Les commandes XSL contenues dans le template peuvent utiliser des expressions Xpath
 - Absolues (racine de l'arbre source)
 - Relatives au noeud courant
- La portée du noeud courant est le template pour lequel il est défini
- Certaines instructions XSL modifient le noeud courant

Instruction xsl:value-of

- Permet de récupérer le contenu des feuilles texte d'un sous arbre :

Feuille de style XSLT

```
<xsl:template match="compte">  
  <p><xsl:value-of select="client"/></p>  
  <p><xsl:value-of select="solde"/></p>  
</xsl:template>
```

Document XML source

```
<compte devise="euro">  
  <numéro>123456KL</numéro>  
  <client>  
    <nom>Dupont</nom>  
    <prénom>Michel</prénom>  
  </client>  
  <solde>1683.25</solde>  
</compte>
```

Résultat

```
<p>DupontMichel</p>  
<p>1683.25</p>
```

Exercice

- **Écrire la feuille de style qui produit la liste des titres des diapositives**
- **Sortie textuelle**

Instruction xsl:apply-templates

- **Appliquer les templates à un ensemble de noeuds**
 - Par défaut : les fils du nœud courant
 - Un ensemble de nœuds désigné par une expression Xpath (attribut select)
- **Apply-templates permet de traiter la descendance d'un nœud auquel est appliqué un template**
 - Par défaut, la descendance d'un noeud auquel est appliqué un template n'est pas traitée

Exemple 1

Feuille de style XSLT

```
<xsl:template match="compte">
  <p>
    Propriétaire du compte :
    <xsl:apply-templates select="client"/>
  </p>
</xsl:template>

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

Document XML source

```
<compte devise="euro">
  <numéro>123456KL</numéro>
  <client>
    <nom>Dupont</nom>
    <prénom>Michel</prénom>
  </client>
  <solde>1683.25</solde>
</compte>
```


Résultat

```
<p>
  Propriétaire du compte :
  <b>Dupont</b>,
  Michel
</p>
```

Exemple 2

Document XML source

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl"
    href="bonjour.xsl"?>
<biblio>
  <book lang="fr">
    <title>XML</title>
    <author>Jean Martin</author>
  </book>
  <book lang="en">
    <title>XML in action</title>
    <author>John Smith</author>
  </book>
  <paper>
    <title>XML software</title>
    <date>05/02/2002</date>
  </paper>
</biblio>
```

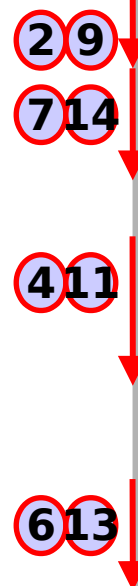


Feuille de style XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns="..."
  <xsl:template match="book">
    <ul>
      <xsl:apply-templates/>
    </ul>
  </xsl:template>

  <xsl:template match="title">
    <li>title...</li>
  </xsl:template>

  <xsl:template match="author">
    <li>author...</li>
  </xsl:template>
</xsl:stylesheet>
```



Résultat de la transformation

```
<ul><li>title...</li>
  <li>author...</li>
</ul>
<ul><li>title...</li>...
```

Exemple 3

Document XML source

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl"
    href="bonjour.xsl"?>
<biblio>
  <book lang="fr">
    <title>XML</title>
    <author>Jean Martin</author>
  </book>
  <book lang="en">
    <title>XML in action</title>
    <author>John Smith</author>
    <publisher ref="3"/>
  </book>
  <editor id="3">
    <name>O'Reilly</name>
    <adress>NewYork</adress>
  </editor>
</biblio>
```

Feuille de style XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns="...">
  <xsl:template match="book">
    < xsl:value-of .../>
    <ul>
      <xsl:apply-templates/>
    </ul>
  </xsl:template>
  <xsl:template match="publisher">
    <apply-templates select=
      "/biblio/editor[@id=@ref]">
  </xsl:template>
  <xsl:template match="editor/name">
    <li>ed : <xsl:value-of.../></li>
  </xsl:template>
</xsl:stylesheet>
```

Résultat de la transformation

```
XML in action
<ul>
<li>ed : O'Reilly</li>
</ul>
```

Exercice

- **Ecrire la feuille de style qui produit un document xhtml valide, contenant la liste des titres de diapositives**
 - Un template pour l'élément *document* source, utilisant apply-templates
 - Un template pour les diapositives

Les origines de XPath

- **Pourquoi ?**
- **Effort d'homogénéisation de la syntaxe et de la sémantique de fonctions communes à**
 - XSLT
 - XPointer
- **Recommandation du W3C (16 novembre 1999)**

Un langage de localisation

- **Objectif principal**

- définir la manière d'adresser les parties d'un document XML
- Utilisation d'une écriture de type « chemin d'accès » pour se déplacer dans la structure d'un arbre représentant le document

- **Objectif secondaire**

- Proposer un ensemble de fonctions prédéfinies pour gérer les expressions booléennes, les nombres et les chaînes de caractères

Le principe de XPath

- **Utilise la même représentation arborescente que XSLT**
- **Xpath permet de définir des expressions s'appliquant à l'arbre source**
- **Une expression Xpath retourne une valeur typée**
 - Ensemble de nœuds (node set)
 - Booléen
 - Nombre
 - Chaîne de caractères

Le principe de XPath

- **Une expression XPath est évaluée dans un contexte**
 - Notion de noeud courant, héritée de XSLT
 - Une paire d'entiers positifs non nuls (la position contextuelle et la dimension contextuelle)
 - Un ensemble de variables (et leurs valeurs) associées au contexte
 - Une bibliothèque de fonctions
 - Un ensemble de déclarations d'espaces de noms

Chemins de localisation

- **Permet de sélectionner un ensemble de nœuds du document**
- **Composé d'une séquence d'étapes séparées par des « / »**
 - Les étapes sont assemblées de gauche à droite
 - Chaque étape sélectionne un ensemble de nœuds qui deviennent à leur tour nœuds contextuels

`/document/body/diapositive/title`

- **Point de départ de l'évaluation**

- le noeud courant (relatif)

`diapositive/title`

- la racine du document (absolu)

`/document/body/diapositive/title`

Chemins de localisation : étapes

- **Une étape d'un chemin de localisation est constitué**

- D'un axe

child | parent | ancestor | descendant | following-sibling |
preceding-sibling | following | preceding | attribute | namespace
| descendant_or_self | ancestor_or_self

- Un nœud de test

nom d'un nœud (avec ou sans préfixe d'espace de nom) ou
« * » ou « text() » ou « comment() » ou « processing
instruction() » ou « node() »

- O ou N **prédicats** pour raffiner l'ensemble des nœuds

Chaque prédicat est exprimé entre crochets droits []

- **Exemple**

child::diapositive[position() = 3]

retourne le 3ème nœud fils de nom « diapositive » du
nœud courant

Chemins de localisation : prédicats

- **Un prédicat filtre un ensemble de nœuds**
 - relativement à un axe de parcours
 - de manière à restreindre cet ensemble
- **Pour chaque nœud de l'ensemble à filtrer, on évalue le prédicat en utilisant**
 - ce nœud comme nœud contextuel,
 - le nombre de nœuds de l'ensemble à filtrer comme dimension contextuelle
 - la position du nœud dans l'ensemble à filtrer comme position contextuelle

Chemins de localisation : prédicats

- **Exécution d'expression de prédicat**
 - évaluation de l'expression
 - conversion du résultat en booléen
- **Si le résultat de l'expression est un nombre :**
 - comparaison (résultat, position du nœud courant)
- **Si le résultat de l'expression n'est pas un nombre :**
 - node-set : est vide renvoie false()
 - chaîne : chaîne nulle renvoie false()

Le type booléen

- **2 valeurs possibles (vrai ou faux)**
- **Opérateurs traditionnels**
and, or, =, <, >
- **Fonctions prédéfinies :**
 - boolean **boolean**(object) convertit ses arguments en booléen
 - boolean **not**(boolean) retourne l'inverse de ses arguments
 - boolean **true**() retourne « true »
 - boolean **false**() retourne « false »
 - boolean **lang**(string) retourne « true » ou « false » selon la langue du nœud contextuel

Les valeurs numériques

- **Nombres flottants**

- Valeur d'un réel, double précision sur 64 bits
- Une valeur spéciale « NaN » (Not a Number)

- **Opérateurs +, -, DIV, MOD**

- **Fonctions prédéfinies :**

- number **number**(object) convertit ses arguments en nombre
- number **sum**(node-set) retourne la somme de la conversion de tous les nœuds passés en argument
- number **floor**(number) retourne le plus grand nombre entier inférieur à l'argument
- number **ceiling**(number) retourne le plus petit nombre entier supérieur à l'argument
- number **round**(number) retourne le nombre entier le plus proche de l'argument

Les chaînes de caractères

- string **string**(object) convertit ses arguments en chaîne de caractères
- string **concat**(string, string*) retourne la concaténation des arguments
- boolean **contains**(string, string) indique si la première chaîne contient la deuxième
- string **substring-before**(string, string) retourne la sous-chaîne du 1er argument qui précède la 1ère occurrence de la deuxième chaîne dans le 1er argument
- string **substring-after**(string, string) inverse de précédemment
- string **substring** (string, number, number) retourne la sous-chaîne du 1er commençant à la position spécifiée par le 2ème argument et sur une longueur spécifiée par le 3ème argument
- string **string-length**(string) retourne le nombre de caractères de la chaîne
- string **normalize-space**(string) retourne la chaîne après avoir normalisé les espaces (suppressions extrémités, fusion séquences)
- string **translate**(string, string, string) retourne la 1ère chaîne dans laquelle toutes les occurrences des caractères de la 2ème chaîne sont remplacées par les caractères de la 3ème chaîne (positions correspondantes)

Les ensembles de noeuds

- number **last**() retourne un nombre égal à la dimension contextuelle
- number **position**() retourne un nombre égal à la position contextuelle
- number **count**(node-set) retourne le nombre de nœuds de l'ensemble
- node-set **id**(object) sélectionne les éléments par leur identifiant unique
 - *id("foo")* sélectionne l'élément qui a pour identifiant unique foo
 - *id("foo")/child::para[position()=5]* sélectionne le 5ème enfant para de l'élément qui a comme identifiant unique foo
- string **local-name**(node-set) retourne la partie locale du nom du 1er nœud de l'ensemble passé en argument
- string **namespace-uri**(node-set) retourne l'URI de l'espace de noms du 1er nœud de l'ensemble passé en argument
- string **name**(node-set) retourne le nom expansé du 1er nœud de l'ensemble passé en argument

Notation abrégée

- **XPath définit des abbréviations**

- *para* est équivalent à *child::para*
- *** sélectionne tous les enfants du nœud courant
- *@nomAttribut* sélectionne l'attribut *nomAttribut*
- *@** sélectionne tous les attributs du nœud courant
- *Para[1]* est équivalent à *child::para[position() = 1]*
- **/para* tous les petits enfants *para* du nœud contextuel
- *chapitre//para* tous les descendants *para* du nœud chapitre
- *chapitre[titre = "Introduction"]*
les enfants *chapitre* du nœud contextuel qui ont au moins un enfant *titre* dont le contenu est égal à *Introduction*
- *para[@type="warning"]*
les enfants *para* dont l'attribut *type* a pour valeur *warning*

Appel de template : apply-templates

- Avec XPath, possibilité de sélectionner précisément les noeuds sur lesquels on applique les templates
- En particulier, pas d'obligation de traiter la descendance
- Lors de l'application d'un template l'attribut select permet de réduire le node-set confronté à l'attribut match du template

```
<xsl:apply-templates select="//para "/>
```

```
<xsl:apply-templates select="para[@class= 'title']"/>
```

```
<xsl:template match="para[1]">
```

```
  ..../..
```

```
</xsl:template>
```

Exercice

- **Écrire la feuille de style produisant, au format xhtml :**
 - La liste des titres des diapositives contenant une image
puis
 - La liste des titres des diapositives ne contenant pas d'image

Appel de template : call-template

- Permet de regrouper dans un template des règles de production utilisées fréquemment

```
<xsl:template match='member'>  
  <xsl:call-template name="info "/>  
  .../  
</xsl:template>
```

```
<xsl:template match='guest'>  
  <xsl:call-template name="info "/>  
  .../  
</xsl:template>
```

```
<xsl:template name='info'>  
  Prénom : <xsl:value-of select="firstname"/>  
  Nom      : <xsl:value-of select="lastname"/>  
</xsl:template>
```

- **Créer des éléments dans le document résultat**
 - Balises n'appartenant pas au namespace XSLT dans les templates : sont reproduites dans la sortie
 - `xsl:element` : génère explicitement un élément
 - `xsl:copy` : copie d'un noeud de l'arbre source
 - `xsl:copy-of` : copie d'un sous-arbre de l'arbre source

Génération d'éléments : xsl:element

- Possibilité de calculer le nom de l'élément
- Possibilité de calculer le namespace (uri)
- Possibilité d'utiliser des groupes d'attributs prédéfinis

```
<xsl:attribute-set name="link-spec">  
    ..../..  
</xsl:attribute-set>
```

```
<xsl:template match="link[@dest='spec']">  
    <xsl:element name="a" use-attribute-sets= "link-spec">  
        <xsl:apply-templates/>  
    </xsl:element>  
</xsl:template>
```

Copie de noeud : xsl:copy

- **Permet de recopier un noeud de la source vers la destination**
 - Copie le noeud courant
 - Ne copie pas les attributs
 - Ne copie pas la descendance
- Possibilité d'utiliser des groupes d'attributs prédéfinis

```
<xsl:template match="@*|node()">
```

```
  <xsl:copy>
```

```
    <xsl:apply-templates select="@*|node()"/>
```

```
  </xsl:copy>
```

```
</xsl:template>
```


Exercice

- **Ecrire la feuille de style identité , produisant un xml identique à l'entrée**
- **Ecrire la feuille de style filtrant les diapositives, en ne gardant que les diapositives utilisant le modèle « description »**
- **Ecrire la feuille de style supprimant les balises « couleur » du document, mais en préservant le contenu**

Copie récursive : xsl:copy-of

- **Permet de copier une arborescence depuis l'arbre source vers le document produit**
- **L'attribut select spécifie le sous-arbre à copier par une expression xpath**
- **Les attributs, namespaces, descendance sont copiés**

```
<xsl:template match="div[@class= 'menu']">  
  <xsl:copy-of select= ". " />  
</xsl:template>
```

Génération d'attributs

- **Attributs des balises littérales**
 - création de l'attribut et de sa valeur
 - utilisation des valeurs d'attributs dynamiques
- **Attribut xslt use-attribute-sets**
 - en conjonction avec xsl:attribute-set
 - sur les balises xsl:copy, xsl:element
- **xsl:copy : copie d'un attribut de l'arbre source**
- **xsl:attribute : génération explicite d'un attribut**

Groupes d'attributs : `xsl:attribute-set`

- **`xsl:attribute-set` définit un groupe d'attributs**
 - Attribut `name` : permet de nommer le groupe d'attributs
 - Contenu : liste de `xsl:attribute`
 - Toujours un élément fils de `xsl:stylesheet`
- **Les éléments `xsl:copy` et `xsl:element` peuvent utiliser les groupes d'attributs**
 - l'attribut `use-attribute-sets` permet de référencer les groupes d'attributs à produire
 - l'attribut `use-attribute-sets` contient une liste de noms de groupes d'attributs séparés par des espaces

Copie d'attributs : xsl:copy

- **Permet de copier un attribut de la source dans la destination**
- **doit figurer avant la génération du contenu de l'élément**
- **Utilisation d'une expression xpath pour désigner les attributs à copier**

```
<xsl:template match="link">  
  <a>  
    <xsl:copy select="./@href"/>  
    <xsl:apply-templates/>  
  </a>  
</xsl:template>
```

Génération d'attributs : xsl:attribute

- **Permet de produire un attribut pour l'élément en cours de production**

- Doit figurer avant la génération du contenu de l'élément
- Attribut name spécifie le nom de l'attribut
- Contenu spécifie la valeur de l'attribut

```
<xsl:template match="link">
```

```
  <a>
```

```
    <xsl:attribute name="title">
```

```
      <xsl:value-of select="linktitle"/>
```

```
    </xsl:attribute>
```

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

```
  </a>
```

```
</xsl:template>
```

Génération de contenu

- **Permet de produire du texte dans le résultat de la transformation**
 - Tout texte contenu dans un template est reproduit
 - L'élément `xsl:text` permet de produire une feuille de texte
 - `xsl:copy` permet de copier du texte
 - Toute feuille texte rencontrée lors du parcours de l'arbre source est implicitement copiée dans le résultat
 - `xsl:value-of` permet de produire une valeur calculée (résultat d'une expression XPath convertie en string)

Génération de contenu : xsl:text

- **Isole le texte, pour ne pas reproduire les retour chariots**

```
<xsl:template match="client">
  <x>Client :
    <xsl:value-of select="name"/>
  </x>
</xsl:template>
```

```
<xsl:template match="client">
  <x>
    <xsl:text>Client : </xsl:text>
    <xsl:value-of select="name"/>
  </x>
</xsl:template>
```

- **Permet de spécifier de ne pas générer d'entités xml**

```
<xsl:text>coût &lt; 100€ </xsl:text>
```

```
<xsl:text disable-output-escaping="yes">coût &lt; 100€ </xsl:text>
```


Exercice

- **Ecrire la feuille de style convertissant l'intégralité de la présentation en document xhtml**
 - Produisant un div avec un attribut class reprenant le modèle pour chaque diapositive
 - Produisant des éléments ul/li pour les listes
 - Produisant les images en xhtml
 - Produisant des éléments en, strong, span à la place des gras, italique, couleur

Boucles : xsl:for-each

- **xsl:for-each** permet de boucler sur les noeuds d'un ensemble de noeuds

- spécifié par une expression xpath (attribut select)
- le noeud courant est modifié à l'intérieur du for-each

```
<xsl:template match="addressbook">  
  <xsl:for-each select="contact/name">  
    <xsl:value-of select="firstname"/>  
    <xsl:text> <xsl:text>  
    <xsl:value-of select="lastname"/>  
  </xsl:for-each>  
</xsl:template>
```

- **Attention :** ne pas reproduire le travail du processeur
 - Ne pas parcourir l'arbre source avec xsl:for-each

Instructions conditionnelles : xsl:if

- **xsl:if permet de traiter une condition**
 - Attribut test : expression xpath évaluée en tant que booléen
 - Il est parfois préférable d'utiliser deux templates

```
<xsl:template match="a">
  <xsl:copy>
    <xsl:attribute name="href">...</xsl:attribute>
    <xsl:if test="not(start-with(@href, 'http://iut2.upmf-grenoble.fr'))">
      <xsl:text>(lien externe)</xsl:text>
    </xsl:if>
    <xsl:apply-templates>
  </xsl:copy>
</xsl:template>
```

Instructions conditionnelles : xsl:choose

- **Conditions multiples**

```
<xsl:choose>
```

```
  <xsl:when test="condition 1">
```

```
    .../...
```

```
  </xsl:when>
```

```
  <xsl:when test="condition 2">
```

```
    .../...
```

```
  </xsl:when>
```

```
  <xsl:otherwise>
```

```
    .../...
```

```
  </xsl:otherwise>
```

```
</xsl:choose>
```

- **xsl:sort permet de spécifier l'ordre d'application d'une instruction sur un node-set**
 - fils de xsl:apply-templates et xsl:for-each
 - Attributs (* : valeur par défaut) :
 - select : critère de tri (expression xpath)
 - lang : pour les tris alphabétiques
 - data-type : text* ou number
 - order : ascending* ou descending
 - case-order : upper-first ou lower-first
 - Si plusieurs éléments xsl:sort sont présent
=> clé de tri primaire, secondaire,...

Exercice

- **Écrire la feuille de style permettant de produire un document xhtml, avec des diapositives triés par ordre alphabétique des titres**

Les variables

- **xsl:variable permet de définir une variable**
 - Non modifiable !
 - Nom de la variable : attribut name
 - Valeur :
 - attribut select (résultat expression xpath)
ou
 - contenu de l'élément xsl:variable (littéral)
- **Les variables peuvent être globales ou locales :**
 - si déclarée dans xsl:stylesheet : variable globale
 - portée : feuille de style
 - si déclarée dans un template : variable locale
 - portée : l'élément contenant la déclaration

Utilisation des variables

- **Les variables peuvent être utilisées dans des expressions Xpath**
- **La valeur d'une variable est obtenue avec le caractère \$**

```
<xsl:variable name="date">27/07/1969</xsl:variable>  
<xsl:variable name="position"  
  select="count(active)"/>
```

```
<xsl:value-of select="$date"/>  
<xsl:apply-templates select="item[$position]"/>
```


Déclaration des paramètres : `xsl:param` (paramètres formel)

- attribut `name` : nom du paramètre
- contenu : valeur par défaut (optionnelle)
- Fils de `xsl:template` (parametre local)
- Fils de `xsl:stylesheet` (parametre global)

Les paramètres effectifs

- **lors de l'appel à la feuille de style**
 - (paramètres globaux)
- **lors de l'appel à des templates**
 - (paramètres locaux)
 - Element **xsl:with-param**
 - Attributs
 - name, select
 - Utilisé dans
 - **xsl:apply-templates**
 - **xsl:call-template**

Passage de paramètres à un template

```
<xsl:stylesheet  ... >
  <!-- parametre global -->
  <xsl:param name='date' />

  <xsl:template match="compte">
    <xsl:apply-templates select="opération">
      <xsl:with-param name="num-compte" select="../@num"/>
      <xsl:with-param name="current-date" select="$date " />
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="opération">
    <xsl:param name="num-compte"/>
    <xsl:param name="current-date">01/01/1972</xsl:param>
    <h3>Opération sur le compte <xsl:value-of select="$num-compte"/></h3>
  </xsl:template>
</xsl:stylesheet>
```

Expressions pour les valeurs d'attributs

- **Facilite la génération d'attributs**
 - Utilisation d'accolades {...} équivalentes à `<xsl:value-of select="..."/>`
 - Peuvent être utilisées dans les valeurs d'attributs produits

Modularité des XSLT

- **Les feuilles de styles sont modularisables :**
 - Inclusion de feuilles de style : `xsl:include`
 - Attribut `href` référence la feuille à inclure
 - Élément fils de `xsl:stylesheet`
 - Import de feuilles de style : `xsl:import`
 - Attribut `href` référence la feuille à inclure
 - Élément fils de `xsl:stylesheet`
 - Les templates des feuilles importées ont une priorité inférieure
 - Appel explicite des templates importés : `xsl:apply-imports`
 - permet d'appliquer des templates importés correspondant au template en cours d'exécution

Exemple xsl:apply-imports

- **doc.xsl**

```
<xsl:template match="example">  
  <pre><xsl:apply-templates/></pre>  
</xsl:template>
```

- **main.xsl**

```
<xsl:import href="doc.xsl"/>  
<xsl:template match="example">  
  <div style="border: solid red">  
    <xsl:apply-imports/>  
  </div>  
</xsl:template>
```

- **résultat**

```
<div style="border: solid red"><pre>...</pre></div>
```

Options de sortie : xsl:output

- **L'élément xsl:output permet de paramétrer les options de sortie de la feuille de style**
 - Élément fils de xsl:stylesheet
 - Attributs :
 - **method** "xml", "text" ou "html"
 - version nmtoken
 - **encoding** string
 - **omit-xml-declaration** "yes" ou "no"
 - standalone "yes" ou "no"
 - doctype-public string
 - doctype-system string
 - cdata-section-elements qnames
 - **indent** "yes" | "no"
 - media-type string

Traitement multi-source

- **Lors du traitement d'un source xml, il est possible :**
 - d'inclure un document xml tiers
 - d'appliquer les templates à un document xml tiers
- Fonction document()
- Utilisée dans une expression XPath

- **Inclusion**

```
<xsl:copy-of select="document('labels.xml')">
```

- **Sous-document**

```
<xsl:variable name='gurl '>
```

```
    http://www.google.fr/search?q=<xsl:value-of select="$keyword"/>
```

```
</xsl:variable>
```

```
<xsl:apply-templates select="document($gurl)//table[2]">
```


Les clés

- **Mécanisme de désignation dans un document**
- **Plus souple que les ID/IDREF de xml**
- **Une clé est définie par :**
 - un nom (attribut name)
 - un ensemble de noeuds (attribut match)
 - pour lesquels la clé est définie
 - spécifié par une expression XPath
 - une valeur pour chaque noeud de l'ensemble (attribut use)
 - une expression XPath évaluée dans le contexte du noeud

Utilisation des clés

- **Dans une expression XPath**
- **A l'aide de la fonction xslt `key(string, object)`**
 - paramètre string : nom de la clé
 - paramètre object : valeur de la clé
 - retourne l'ensemble des noeuds pour lesquels
 - une clé ayant le nom fourni est définie
 - la valeur de cette clé est la valeur fournie

Les messages

- **Possibilité de produire des messages au cours de la transformations**
 - sur la console (transformation batch)
 - dans un fichier de log (processeur sur serveur http)
 - dans un pop-up (processeur dans navigateur ou environnement auteur)

```
<xsl:template match="section">
  <xsl:message>
    <xsl:text>Traitement de la section</xsl:text>
    <xsl:value-of select="title"/>
  </xsl:message>
  ../..
</xsl:template>
```

Les modes

- Permet de regrouper les templates pour définir des traitements particuliers
- Attribut *mode* sur *xsl:template* et *xsl:apply-templates*

```
<xsl:template match="chapter">
  <div class="sommaire"><ul>
    <xsl:apply-templates select="section" mode="sommaire"/>
  </ul></div>
  <xsl:apply-templates select="section"/>
</xsl:template>
```

```
<xsl:template match="section" mode="sommaire"/>
  <li><xsl:value-of select="title"/></li>
</xsl:template>
```

```
<xsl:template match="section">
  <div class="section"><xsl:apply-templates/></div>
</xsl:template>
```

Exercice

- **Ecrire la feuille de style produisant une présentation xhtml, incluant la liste des titres de diapositives au début du fichier**
- **Enrichir la feuille précédente en plaçant des liens vers les diapositives depuis le sommaire**

XSLT et les namespaces

- **Si le document source, ou le document produit utilise plusieurs espaces de nommage,**
 - l'ensemble de ceux-ci doit être déclaré dans la feuille de style
 - Ils doivent être utilisés dans les balises produites et les Xpath

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

  .../...

  <xsl:template match="html:body">
    <fo:flow>
      .../...
    </fo:flow>
  </xsl:template>

</xsl:stylesheet>
```