

# La spécification des chemins avec XPath XSLT avancé

L3Pro BDISE – XML

Mathieu Sassolas

IUT de Sénart Fontainebleau  
Département Informatique

Année 2016-2017  
Cours 4



- 1 La spécification des chemins avec XPath
  - Principe général
  - Les axes
  - Les filtres
  - Les expressions pour les tests et prédicats
  - Mise en application
- 2 Quelques fonctionnalités avancées de XSLT
  - Tri
  - Variables et paramètres
  - Petites astuces
  - Mise en application

- ▶ Spécification des nœuds sélectionnés pour appliquer des templates ou récupérer une valeur lors des `select` (ou des `match` dans la définition des templates).
- ▶ Langage pour les conditions des test (`xsl:when`, `xsl:if...`).

- ▶ Spécification des nœuds sélectionnés pour appliquer des templates ou récupérer une valeur lors des `select` (ou des `match` dans la définition des templates).
  - ▶ Langage pour les conditions des test (`xsl:when`, `xsl:if...`).
- ↪ Parcours dans l'arbre avec des tests sur le chemin.

- ▶ Un chemin est une suite d'étapes :

## Syntaxe d'un chemin

```
[/]etape1/etape2/.../etapeN
```

- ▶ Dans chaque étape, on navigue dans l'arbre selon des **axes** : père, fils, frère...
- ▶ On peut également **filtrer** selon le **type de nœud**.
- ▶ On peut enfin demander que des **prédicats** soient vérifiés.

## Syntaxe d'une étape

```
axe::filtre [predicat1] [predicat2] ...
```

~>> C'est juste de l'anglais !

`self` Le nœud courant.

`attribute` Les attributs du nœud courant.

`child` Les fils.

`descendant` Les descendants (fils, petit-fils,...).

`descendant-or-self` Les descendants ou le nœud courant.

`parent` Le père.

`ancestor` Les ancêtres (père, grand-père,...) ; en particulier la racine est toujours un tel nœud.

`ancestor-or-self` Les ancêtres ou le nœud courant.

**following-sibling** Les frères suivants.

**preceding-sibling** Les frères précédents.

**following** Les nœuds suivants, frères ou non (suivant dans la lecture XML).

**preceding** Les nœuds précédents, frères ou non (précédant dans la lecture XML).

**namespace** Les nœuds ayant le même espace de nom que le nœud courant.

Pour XPath, tout est un **nœud** (**node**), mais il y en a plusieurs types : attributs, éléments, texte, commentaire (principalement).

- ▶ Tous les nœuds : **node()**.
- ▶ Tous les types d'attributs : **\*** (lorsque l'on navigue dans l'axe **attribut**)
- ▶ Tous les types d'éléments : **\*** (sinon)
- ▶ Un type d'**élément** particulier : **MonElement**.
- ▶ Les nœuds de type texte : **text()**.
- ▶ Les nœuds de type commentaire : **comment()**.

`child::monElement` Tous les fils de type `monElement` (équivalent abrégé : `monElement`).

`parent::node()/attribute::*` Tous les attributs du nœud père (équivalent abrégé : `../@*`).

`descendant-or-self::comment()` Tous les commentaires se trouvant dans les descendants au sens large (équivalent abrégé : `//comment()`).

`descendant::text()` Tous le texte (càd #PCDATA) dans les descendants stricts (équivalent abrégé : `*//text()`).

↪ Servent dans les `test="..."` et dans les prédicats :  
`axe::filtre[prédicat]`.

`elt` Présence d'un élément fils de type `elt` (peut être un chemin XPath plus complexe...).

`@attribut` Présence d'un attribut de type attribut.

`@attribut = 'texte'` Teste si la valeur de l'attribut attribut est égale à au texte texte.

`@attribut != 'texte'` Teste si la valeur de l'attribut attribut est différente du texte texte.

`@attribut <op> valeur` où `<op> ∈ {&lt;; &lt;=, &gt;; &gt;=, =, !=}` et valeur est un entier.

### Remarque

*On peut tester également la valeur d'un élément, mais ça ne prend pas que la partie texte : il faut utiliser `element/texte()`.*

↪ On peut manipuler un peu plus les valeurs d'attribut (et d'éléments) via des **fonctions**.

- ▶ Opérations arithmétiques (valeurs numériques) : **+**, **-**, **\***, **div** (division flottante), **mod** (reste de la division euclidienne).
- ▶ Opérations booléennes : **and**, **or**, **not(...)**.
- ▶ Opérations de traduction : **translate(texte, caractère\_à\_replacer, remplacer\_par)**  
 ↪ Le remplacement fonctionne **caractère par caractère** et est sensible à la casse !

### Exemple

```
translate('Abracadabra', 'ba', 'zo') ↪ Azrocodozro
translate('Abracadabra', 'ba', 'z') ↪ Azrcdizr
```

- ▶ Le nœud courant : `current()`.

**Attention !**

N'est **pas** relatif au chemin XPath en cours d'évaluation !

- ▶ Le nœud courant : `current()`.

```

<xsl:template match="pere">
Current: <xsl:value-of
select="fils[current()/@prefere = @code]/@nom"/>
Self ou équivalents:
self::node() <xsl:value-of
select="fils[self::node()/@prefere = @code]/@nom"/>
. <xsl:value-of select="fils[./@prefere = @code]/@nom"/>
&apos;rien&apos;;
<xsl:value-of select="fils[@prefere = @code]/@nom"/>
</xsl:template>
  
```

```
<pere prefere="GG" code="PP">  
<fils code="DD" nom="Dédé" prefere="PP"/>  
<fils code="GG" nom="Gégé" prefere="DD"/>  
<fils code="TT" nom="Toto" prefere="TT"/>  
</pere>
```

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

```
Current: <xsl:value-of
```

```
select="fils[current()/@prefere = @code]/@nom"/>
```

```
Self ou équivalents:
```

```
self::node() <xsl:value-of
```

```
select="fils[self::node()/@prefere = @code]/@nom"/>
```

```
. <xsl:value-of select="fils[./@prefere = @code]/@nom"/>
```

```
&apos;rien&apos;
```

```
<xsl:value-of select="fils[@prefere = @code]/@nom"/>
```

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

```
<pere prefere="GG" code="PP">  
<fils code="DD" nom="Dédé" prefere="PP"/>  
<fils code="GG" nom="Gégé" prefere="DD"/>  
<fils code="TT" nom="Toto" prefere="TT"/>  
</pere>
```

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

```
Current: <xsl:value-of
```

```
select="fils[current()/@prefere = @code]/@nom"/>
```

```
Self ou équivalents:
```

```
self::node() <xsl:value-of
```

```
select="fils[self::node()/@prefere = @code]/@nom"/>
```

```
. <xsl:value-of select="fils[./@prefere = @code]/@nom"/>
```

```
&apos;rien&apos;
```

```
<xsl:value-of select="fils[@prefere = @code]/@nom"/>
```

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

```
Current: Gégé;gégé;
```

```
Self ou équivalents:
```

```
self::node() Toto
```

```
. Toto
```

```
'rien' Toto
```

- ▶ Le nœud courant : `current()`.

### Attention !

N'est **pas** relatif au chemin XPath en cours d'évaluation !

- ▶ Union d'un ensemble de nœuds : `|`, par exemple  
`elt[fils1 | fils2]/petitfils.`

- ▶ Le nœud courant : `current()`.

### Attention !

N'est **pas** relatif au chemin XPath en cours d'évaluation !

- ▶ Union d'un ensemble de nœuds : `|`, par exemple `elt[fils1 | fils2]/petitfils`.
- ▶ Tests sur la position du nœud parmi ses frères :  
`position() = last()`, `position() = 1`, `position() mod 2 = 0`.

### Remarque

*Attention, les nœuds de texte, de commentaires, etc sont aussi comptés. À n'utiliser que dans un ensemble de nœud que l'on connaît : `elt[position()=last()]` choisit le dernier **parmi les éléments** `elt`.*

- ▶ Sur les chaînes de caractères :
  - `contains(contenant, contenu)` Si le contenant contient le contenu. **Sensible à la casse!**
  - `concat(chaine1, chaine2, ...)` Concaténation de chaînes.
  - `substring(chaine, pos, longueur)` Sous-chaîne de longueur long à partir de la position pos.
- ▶ Fonctions numériques :
  - `count(chemin/Xpath)` Compte le nombre de nœuds désignés par le chemin XPath.
  - `sum(chemin/Xpath)` Fait la somme des valeurs des nœuds désignés par le chemin XPath (n'a du sens que si le contenu du chemin sont des nombres).

- ▶ Il existe d'autres fonctions que celles présentées.

## Attention

On utilise XSLT 1.0, des fonctions trouvées dans des docs pourraient ne fonctionner que dans XSLT 2.0.

- ▶ Il existe d'autres fonctions que celles présentées.

## Attention

On utilise XSLT 1.0, des fonctions trouvées dans des docs pourraient ne fonctionner que dans XSLT 2.0.

- ▶ Les prédicats s'évaluent dans le contexte **local** :  
`pere/fils[@attr="42"]` teste la valeur de l'attribut `attr` du **fils**.
- ▶ Les **enchaînement de prédicats** ne fonctionnent pas exactement comme des conjonction : `elt [pred1] [pred2]` teste `pred2` sur les nœuds désignés par `elt [pred1]`  
↪ Attention en particulier lors de tests avec `position()`.

- ▶ Il existe d'autres fonctions que celles présentées.

Enchaînement:

```
<xsl:value-of  
select="viande[@gras = 'no'] [position() = 1]/@id"/>  
[end]
```

Conjonction:

```
<xsl:value-of  
select="viande[@gras = 'no' and position() = 1]/@id"/>  
[end]
```

- ▶ Les enchaînement de prédicats

exactement comme des conjonction : `elt [pred1] [pred2]`  
teste `pred2` sur les nœuds désignés par `elt [pred1]`

↪ Attention en particulier lors de tests avec `position()`.

```
<viande gras="yes" id="1"/>
<viande gras="no" id="2"/>
<viande gras="no" id="3"/>
<viande gras="yes" id="4"/>
```

- ▶ Il existe d'autres fonctions que celles présentées.

Enchaînement:

```
<xsl:value-of
select="viande[@gras = 'no'] [position() = 1]/@id"/>
[end]
```

Conjonction:

```
<xsl:value-of
select="viande[@gras = 'no' and position() = 1]/@id"/>
[end]
```

- ▶ Les enchaînement de prédicats

exactement : elt [pred1] [pred2]  
 teste par 2 [end] par elt [pred1]  
 ⚡ Attention : tests avec position().  
 Conjonction:  
 [end]

XPath &  
XSLT avancé

M. Sassolas

L3Pro

Cours 4

XPath

Principe

Axes

Filtres

Expressions

Mise en application

XSLT avancé

↳ C'est l'heure du TD sur XPath ←

XPath &  
XSLT avancé

M. Sassolas

L3Pro

Cours 4

XPath

Principe

Axes

Filtres

Expressions

Mise en application

XSLT avancé

## Syntaxe (abrégée)

```
<xsl:sort select="champ_pour_trier"  
           [order="ascending|descending"  
           [data-type="text|number"]/>
```

- ▶ Au sein d'une boucle `for-each` ou d'un `apply-templates`.
- ▶ Le champ pour trier est un chemin XPath **relatif aux nœuds sélectionnés** par le `for-each` ou l'`apply-templates`.
- ▶ On peut les enchaîner : on trie d'abord selon le premier critère puis selon le second en cas d'égalité, etc.

## Syntaxe (abrégée)

```
<xsl:apply-templates>  
<xsl:sort select="@gras"/>  
<xsl:sort select="@note" data-type="number"/>  
</xsl:apply-templates>  
...  
<xsl:template match="viande">  
<xsl:value-of select="@id"/>;  
</xsl:template>
```

- ▶ On peut les enchaîner : on trie d'abord selon le premier critère puis selon le second en cas d'égalité, etc.

```
T <viande gras="yes" id="1" note="C"/>  
<viande gras="no" id="2" note="B"/>  
<viande gras="no" id="3" note="A"/>  
<viande gras="yes" id="4" note="B"/>  
S <viande gras="maybe" id="5" note="A"/>
```

```
<xsl:apply-templates>  
<xsl:sort select="@gras"/>  
<xsl:sort select="@note" data-type="number"/>  
</xsl:apply-templates>  
...  
<xsl:template match="viande">  
<xsl:value-of select="@id"/>;  
</xsl:template>
```

- On peut les enchaîner : on trie d'abord selon le premier critère puis selon le second en cas d'égalité, etc.

```
T
<viande gras="yes" id="1" note="C"/>
<viande gras="no" id="2" note="B"/>
<viande gras="no" id="3" note="A"/>
<viande gras="yes" id="4" note="B"/>
<viande gras="maybe" id="5" note="A"/>
```

```
<xsl:apply-templates>
<xsl:sort select="@gras"/>
<xsl:sort select="@note" data-type="number"/>
</xsl:apply-templates>
...
<xsl:template match="viande">
<xsl:value-of select="@id"/>;
</xsl:template>
```

```
5;
3;
2;
4;
1;
```

- On peut les encadrer dans un `<xsl:sort data-type="number"/>` pour trier d'abord selon le premier critère puis selon le second, etc.

## Syntaxe

```
<xsl:variable name="nomVariable">  
  Du contenu <tag>potentiellement</tag> XML  
</xsl:variable>  
  
...  
<xsl:value-of select="$nomVariable"/>  
<xsl:copy-of select="$nomVariable"/>
```

- ↪ On ne peut pas **réaffecter** ni **redéfinir** une variable.
- ▶ `value-of` prend la valeur textuelle de la variable.
  - ▶ `copy-of` prend la valeur tout le contenu de la variable, y compris le XML.

## Syntaxe

```
<xsl:variable name="nomVariable">  
  Du contenu <tag>potentiellement</tag> XML  
</xsl:variable>  
...  
<xsl:value-of select="$nomVariable"/>  
<xsl:copy-of select="$nomVariable"/>
```

- ↪ On ne peut pas **réaffecter** ni **redéfinir** une variable.
- ▶ `value-of` prend la valeur textuelle de la variable.
  - ▶ `copy-of` prend la valeur tout le contenu de la variable, y

Du contenu potentiellement XML

Du contenu <tag>potentiellement</tag> XML

## Syntaxe

```
<xsl:variable name="nomVariable">
  Du contenu <tag>potentiellement</tag> XML
</xsl:variable>
...
<xsl:value-of select="$nomVariable"/>
<xsl:copy-of select="$nomVariable"/>
```

- ↪ On ne peut pas **réaffecter** ni **redéfinir** une variable.
- ▶ `value-of` prend la valeur textuelle de la variable.
  - ▶ `copy-of` prend la valeur tout le contenu de la variable, y compris le XML.

La **portée** de la variable est tout son parent `xsl:...`, après sa déclaration. Cela comprend donc les descendants de ses frères.

↪ Le contenu des variables peut aussi provenir du document via XSLT.

## Exemple

```
<xsl:variable name="sexe">  
  <xsl:choose>  
    <xsl:when test="@genre = 'M'">  
      C'est un homme</xsl:when>  
    <xsl:when test="@genre = 'F'">  
      C'est une femme</xsl:when>  
    <xsl:otherwise>On ne sait pas:  
  "<xsl:value-of select="@genre"/>"</xsl:otherwise>  
  </xsl:choose>  
</xsl:variable>
```

- ▶ Les variables sont locales, elles ne peuvent être passées en paramètre dans d'autres templates.
- ▶ Les paramètres, au contraire, sont faits pour ça.

## Syntaxe

```
<xsl:template name="nomTemplate">
  <xsl:param name="nomParametre"
              select="'valeurParDefaut'"/>
  ...
</xsl:template>
...
<xsl:call-template name="nomTemplate">
  <xsl:with-param name="nomParametre"
                  select="valeur/qui [peut/etre] /unchemin/@Xpath"/>
</xsl:call-template/>
```

## Syntaxe

```
<xsl:comment>  
  Du commentaire  
</xsl:comment>
```

## Exemple

```
<xsl:template match="comment()">  
  <xsl:comment>  
    Commentaire repris du XML original:  
    "" ""  
    <xsl:value-of select="."/>  
    "" ""  
  </xsl:comment>  
</xsl:template>
```

## Syntaxe

```
<xsl:variable name="nomVariable">
  Du contenu <tag>potentiellement</tag> XML
</xsl:variable>
...
<xsl:copy>Nouveau contenu</xsl:copy>
<xsl:copy-of select="element/ou/chemin"/>
```

- ▶ Deux manières de copier.
- ▶ Copie de l'**élément** sans copie des fils (ni des **attributs**).
- ▶ **Copie profonde**, avec les descendants.
- ▶ `xsl:copy-of` doit **toujours** être un élément vide.

### Document XML

```
<elem attr="intro">Bonjour</elem>
```

### Feuille XSLT

Shallow: `<xsl:copy>Au revoir</xsl:copy>`

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

### Résultat

Shallow: `<elem>Au revoir</elem>`

Deep: `<elem attr="intro">Bonjour</elem>`

## Syntaxe

```
<xsl:text>  
  Du texte sans balises!  
</xsl:comment>
```

## Exemple (XSLT)

```
<xsl:text>Avant un truc: </xsl:text>  
<xsl:value-of select="@note"/>  
<xsl:text>. Après le truc.</xsl:text>
```

## Exemple (Résultat)

Avant un truc: A. Aprè&grave; s le truc.

## Remarque

*Les espaces autour de la valeur sont corrects !*

## Utiliser le navigateur comme processeur XSLT.

- ▶ **Fortement déconseillé** lors de la phase d'écriture de la feuille XSL : **débuggage** impossible.
- ▶ Plutôt utile pour rendre facile la lecture de XML (transformation en HTML plutôt qu'ajout de CSS).
- ▶ Demande d'avoir la main sur le XML.

### Commande

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

XPath &  
XSLT avancé

M. Sassolas

L3Pro

Cours 4

XPath

XSLT avancé

Tri

Variables et  
paramètres

Petites astuces

Mise en application

↳ C'est l'heure du TP ←

XPath &  
XSLT avancé

M. Sassolas

L3Pro

Cours 4

XPath

XSLT avancé

Tri

Variables et  
paramètres

Petites astuces

Mise en application