# Eléments de programmation XSLT

**Code: xml-xslt2**

## Originaux

*url:* **http://tecfa.unige.ch/guides/tie/html/xml-xslt/xml-xslt2.html**

*url:* **http://tecfa.unige.ch/guides/tie/pdf/files/xml-xslt2.pdf**

## Auteurs et version


- Daniel K. Schneider - Vivian Synteta

- Version: 0.4 (modifé le 2/12/05 par DKS)

## Prérequis

*Module technique précédent:* xml-dom (éléments du XML Framework)

*Module technique précédent:* xml-tech (arbres XML, DTDs)

*Module technique précédent:* xml-xpath (Important !!)

*Module d'exercices précédant:* xml-xslt (bases !!)

## Modules suivants

*Module technique suivant:* xml-xslfo

# Objectifs

- XSLT 1.0 moyen (éléments de programmation)

# Notes:

- Brouillon pour le moment (il manque des éléments XSLT)
- Limité à XSLT 1.0 (il existe déjà des processeurs pour XSLT 2.0)

# 1. Table des matières détaillée

# 2. Les expressions et fonctions XPath

- XSLT utilise le langage d'expressions défini par XPath [XPath]. Les expressions sont utilisées dans XSLT pour une variété de possibilités incluant :
  - sélection de noeuds pour traitement;
  - spécification des conditions pour les différentes manières de traitement d'un noeud;
  - génération de texte à insérer dans l'arbre résultat.

**Consultez donc aussi:**

*url:* http://tecfa.unige.ch/guides/tie/html/xml-xpath/xml-xpath.html

- Sinon, on introduira quelques fonctions dans les exemples par la suite ...

**Petit exemple d'utilisation**

```
<xsl:if test="position() !=last()">, </xsl:if>
```

Traduction: Si la position de l'élément (noeud) courant n'est pas égale à la dernière dans la liste, alors insérer une virgule.

**Quelques explications**

- le contexte pour l'application d'une fonction XPath est toujours le noeud courant.
- Donc `position()` indique la position d'un noeud par rapport aux "frères" imbriqués dans le même parent (appelés aussi "liste courante de noeuds")
- `last()` indique s'il s'agit du dernier élément
- Attention: la première position est 1 !! (et pas zéro comme dans la plupart des langages)

# 3. La logique de la programmation par règles

## 3.1 Rappel: Utilisation de apply-templates et XPath

- Avant d'utiliser des constructions avancées comme "if" ou "for-each", réfléchissez bien si c'est vraiment nécessaire. Dans la plupart des cas il suffit de définir des règles avec "xsl:apply-templates" et le reste "s'organise" tout seul.
- "Lay back and let the rules do the work ...."

### Exemple 3-1: Simple XML vers HTML avec XSLT

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-simple/ (fouiller le répertoire !)**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-simple/simple-xpath-apply.xml**

## A. Un texte en XML

```
<arbre>
 <para>Simples Templates et XPath</para>
 <aunt>
  <name>Auntie</name>
  <child>Je suis un enfant de aunt</child>
 </aunt>
 <uncle>
  <name>Uncle Ben</name>
  <child>Je suis le premier enfant de uncle</child>
  <child>Je suis le 2eme enfant de uncle</child>
  <child>Je suis le 3eme enfant de uncle</child>
 </uncle>
</arbre>
```

# B. La feuille de style XSLT

- Ici on veut produire un simple HTML à partir du XML
- On aimerait que les enfants de <aunt> et <uncle> soient affichés différemment
- Le premier enfant de <uncle> doit être affiché spécialement aussi
- Si voulez savoir comment ce code s'exécute, consultez le fichier *-trace.xml

```
<xsl:template match="arbre">
 <html><title>XSL Example</title><body>
   <xsl:apply-templates />
  </body> </html>
</xsl:template>

<xsl:template match="uncle|aunt"> <hr /> <xsl:apply-templates />  </xsl:template>

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

<xsl:template match="uncle/child[position()=1]">
 <p> <strong><xsl:apply-templates /></strong> </p>
</xsl:template>

<xsl:template match="uncle/child[position()>1]">
 <p> <xsl:apply-templates /></p>
</xsl:template>

<xsl:template match="aunt/child">
 <p style="color:blue"><xsl:apply-templates /></p>
</xsl:template>
```

# C. Les résultat en HTML

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <title>XSL Example</title>
   <body>
      Simples Templates et XPath
      <hr>
      Auntie :
      <p style="color:blue">Je suis un enfant de aunt</p>
      <hr>
      Uncle Ben :
      <p><strong>Je suis le premier enfant de uncle</strong></p>
      <p>Je suis le 2eme enfant de uncle</p>
      <p>Je suis le 3eme enfant de uncle</p>
   </body>
</html>
```

# D. Rappel du principe

- au lieu de programmer des boucles, des if/else, appels de fonction etc. comme on le verra dans la suite, on définit plutôt des règles qui se déclenchent en fonction de la position d'un élément (ou encore de leurs attributs, contenus, etc.)

# 3.2 Les règles implicites

- Les règles implicites sont souvent source de confusion ! Lisez cette section si dans votre arbre de sortie apparaît du texte que vous n'avez pas voulu inclure ....

## Le "problème"

- Citation de la spécification:
  - "Il existe une règle modèle [template] interne permettant à un traitement récursif de continuer même en cas de non-concordance de motif par une règle modèle explicite de la feuille de styles. Cette règle modèle s'applique aussi bien aux noeuds d'éléments qu'au noeud racine. Ce qui suit montre ce qui équivaut à cette règle interne" :

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

  - Il existe aussi un template interne pour les noeuds textuels et les noeuds d'attributs qui sert à en recopier le texte :

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

- En clair cela veut dire que si vous ne spécifiez pas explicitement une règle pour les tous les éléments de votre arbre XML, XSLT va appliquer deux règles prédéfinies et qui vont copier le contenu des éléments sans règle.
- Autrement dit, si vous ne voulez pas que le texte de ces éléments et attributs apparaisse, il faut agir ... !!

## Solution

- Soit redéfinir ces 2 règles,

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

  - notez la balise auto-fermante "/>".
  - il s'agit vraiment de règles qui ne produisent rien ....

- Soit définir une règle pour chaque élément qui ne produit rien:

```
<xsl:template match="element_qui_ne_sert_a_rien" />
```

  - Cette 2ème solution est préférable, car souvent on utilise la règle implicite pour juste insérer le texte d'un noeud.
  - Au lieu d'écrire:

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

  - On utilise

```
<xsl:template match="title">
  <p><xsl:apply-templates></p>
</xsl:template>
```

# 4. Eléments de programmation

## 4.1 Traitement conditionnel par xsl:if

***Syntaxe:***

```
<xsl:if  test = "boolean-expression">
 <!-- Content: template -->
</xsl:if>
```

- Permet de changer l'output en fonction d'un test
- Attention: il n'existe pas de "else" (utilisez "choose" à la place)
- Cas d'utilisation: Traitement d'un élément en fonction de sa position, des ses attributs etc.

### Exemple 4-1: xsl:if exemple pour insérer des virgules dans une liste

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-if/simple-comma-list.xml**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-if/simple-comma-list.xsl**

```
<example>
 <list>
  <element>element A</element>  <element>element B</element>
  <element>element C</element>  <element>element D</element>
  <element>element E</element>
 </list>
 <list>
  <element>element AA</element> <element>element BB</element>
  ...
</element>
 </list>
```

## Résultat affiché:

```
Liste: element A, element B, element C, element D, element E.
Liste: element AA, element BB, element CC, .
```

## Feuille de style

```
<xsl:template match="example">
 <html><title> <xsl:value-of select="title"/></title>
  <body> <xsl:apply-templates/> </body>
 </html>
</xsl:template>

<xsl:template match="list">
 Liste:
 <xsl:apply-templates/>.
 <br/>
</xsl:template>

<xsl:template match="element">
 <xsl:apply-templates/>
 <xsl:if test="position()!=last()"> <xsl:text>, </xsl:text> </xsl:if>
</xsl:template>
```

## Problème:

- Différence de modèle de traitement entre Microsoft et autres engins XSLT
- Le dernier élément vide "reçoit" aussi une virgule dans Firefox par exemple
- Solution: Soit serrer" les balises, soit virer les "whitespaces"
- Cf. 5.1 “Génération conditionnelle de texte” [17]

# 4.2 Traitement conditionnel avec xsl:choose

- une forme plus complexe que "if"

***Syntaxe:***

```
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>

<xsl:when
  test = boolean-expression>
  <!-- Content: template -->
</xsl:when>

<xsl:otherwise>
  <!-- Content: template -->
</xsl:otherwise>
```

**Cette définition dit:**

- on peut avoir plusieurs clauses avec un test (xsl:when).
- la première vraie est utilisée
  - donc la série des xsl:when correspond à "if ( ) { ...}" elseif ( ) { ...}" elseif ( ) { ...}" de PHP
- Si aucune clause n'est vraie et s'il existe une clause xsl:otherwise, c'est cette dernière qui est exécutée
  - il s'agit donc du "else {..}"

## Exemple 4-2: Couleurs pour animaux colorés

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-choose/animal-eyes.xml**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-choose/animal-eyes.xsl**

## Le XML:

```
<example>
 <title>Simple XSLT example that will show how to deal with animal eyes
 </title>
 <list>
  <animal couleur="noir">Panthère</animal>
  <animal couleur="compliquée avec des tâches">Panthère</animal>
  <animal couleur="blanc">Ours polaire</animal>
  <animal couleur="vert">Grenouille</animal>
  <animal>Vache</animal>
 </list>
```

- Chaque animal sera listé
- La couleur du texte pour certains animaux sera celle de sa couleur

## Le XSLT:

```
 <xsl:template match="list">
  Liste d'animaux:
  <ul>   <xsl:apply-templates/>    </ul>
 </xsl:template>

<xsl:template match="animal">
  <li>
  <xsl:choose>
    <xsl:when test="@couleur='noir'">
      <p style="color:black;font-weight:bold">
        <xsl:value-of select="."/>
      </p>
    </xsl:when>
    <xsl:when test="@couleur='vert'">
      <p style="color:green">
        <xsl:value-of select="."/>
      </p>
    </xsl:when>
    <xsl:otherwise>
      <p style="color:blue">
        <xsl:value-of select="."/>
      </p>
    </xsl:otherwise>
  </xsl:choose>
  </li>
</xsl:template>
```

# 4.3 Boucles xsl:for-each

- Cas d'utilisation: faire des tables

## Exemple 4-3: Présentation du résultat XML d'une sortie SQL typique avec XSLT

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-foreach/rowset.xml**
*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-foreach/rowset.xsl**

## Extrait du fichier XML

```
<ROWSET>
   <ROW>
    <id>1</id>
    <login>test</login>
    <fullname>Joe Test </fullname>
    <url>http://tecfa.unige.ch/</url>
    <food>3</food>
    <work>4</work>
    <love>5</love>
    <leisure>2</leisure>
   </ROW>

   <ROW>
     .....
   </ROW>
   ....
</ROWSET>
```

# Extrait du fichier XSLT

```
<xsl:template match="ROWSET">
    <table border="2" cellspacing="1" cellpadding="6">
    <tr><th>id</th>
        <th>Login</th>
        <th>Full Name</th>
        <th>URL</th>
        <th>food</th><th>work</th><th>love</th><th>leisure</th>
     </tr>
    <xsl:for-each select="ROW">
     <tr>
      <td><xsl:value-of select="id"/></td>
      <td><xsl:value-of select="login"/></td>
      <td><xsl:value-of select="fullname"/></td>
        <td bgcolor="tan"><a href="{url}"><xsl:value-of select="url"/></a></td>
      <td><xsl:value-of select="food"/></td>
      <td><xsl:value-of select="work"/></td>
      <td><xsl:value-of select="love"/></td>
      <td><xsl:value-of select="leisure"/></td>
      </tr>
     </xsl:for-each>
    </table>
</xsl:template>
```

- **xsl:for-each select="ROW"**
  - extrait pour chaque "ligne" de la sortie SQL (ROW) les valeurs qui nous intéressent
  - met des balises "<tr>" au début à la fin d'un passage
  - met des balises "<td>

# 5. Exemples simples

## 5.1 Génération conditionnelle de texte

### Exemple 5-1: Insertion de virgules dans une liste à longueur variable

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-if/**

Le problème:

- Afficher le contenu d'un sous-élément (c.à.d. d'une liste) en fonction de sa position.
- Ici on montre comment insérer des virgules entre les éléments, mais pas après le dernier.
- On utilise la fonction position() du langage XPath pour déterminer la position d'un élément dans son contexte (c.à.d. l'élément mère)

### Première opération: gérer les éléments de type "text node" vide

- problème: pour les processeurs XPath/XSLT les espaces, retours de lignes, tabs etc. situés ***autour*** des éléments sont aussi des noeuds qu'il va compter. La solution simple consiste à éliminer ces espaces:

```
<list><!-- ici va se mettre un noeud (node) vide -->
  <element>element AA</element> <!-- ici un node vide ... -->
  <element>element BB</element>
  <element>element CC</element><!-- ici un node vide ... -->
 </list>
Donc:
<xsl:strip-space elements="list"/>
```

## Insérer les virgules avec un test:

Voir les fichiers simple-list.xml et simple-list.xsl

```
<xsl:template match="list">
 Liste: <xsl:apply-templates/>. <br/>
</xsl:template>

<xsl:template match="element">
 <xsl:apply-templates/>
 [pos=<xsl:value-of select="position()"/>]
 <xsl:if test="position()!=last()"> <xsl:text>, </xsl:text> </xsl:if>
</xsl:template>
```

Résultat affiché

```
Liste: element AA [pos=1] , element BB [pos=2] , element CC [pos=3] .
```

## Même opération lorsque les "enfants" sont des éléments différents:

Voir les fichiers reference-entry.xml et reference-entry.xsl

```
<xsl:template match="ref">
 <xsl:apply-templates
select="author|ref|title|edition|publisher|pubPlace|publicationYear"/>.
</xsl:template>

<xsl:template match="author|edition|publisher|pubPlace|publicationYear">
 <xsl:apply-templates/>
 <xsl:if test="position() !=last()">, </xsl:if>
</xsl:template>
```

# 5.2 Fabrication de références (liens)

## Exemple 5-2: Table des matières pour éléments qui ont un identificateur

*url:* **http://tecfa.unige.ch/guides/xml/examples/recit/**

## Fragment XSLT (pour faire les liens au départ, voir: mode="toc")

```
<xsl:template match="/">
  <html>
   <body bgcolor="#FFFFFF">
     <h1><xsl:value-of select="/RECIT/Titre"/></h1>
     <p> Highlights de l'histoire:
         <xsl:apply-templates select="//EPISODE" mode="toc"/> </p>
     <xsl:apply-templates/>
   </body>
 </html>
</xsl:template>
```

## Fragments XSLT (pour gérér la balise "EPISODE")

```
<xsl:template match="EPISODE" mode="toc">
  <a href="#{@id}"><xsl:value-of select="SousBut"/></a> -
</xsl:template>

<xsl:template match="/RECIT/FIL/EPISODE">
  <a name="{@id}"> <hr /> </a>
  <xsl:apply-templates/>
</xsl:template>
```

## Exemple 5-3: Tables des matières pour éléments sans ID

- plus difficile car il faut "fabriquer" des attributs "name" et "href" pour le HTML
  - la solution adoptée ici est moche (on aurait pu compter les éléments)
- il s'agit des jours d'un atelier webmaster qu'on peut consulter ici:

  *url:* **http://tecfa.unige.ch/tecfa/teaching/formcont/webmaster2003/programme/**
- Pour comprendre cet exemple il faut fouiller dans les fichiers suivants:
  - programme.xml - contient tout le "programme" de l'Atelier
  - programme.xsl - produit le résultat (voir le fichier programme.html)
  - matos.xsl, resume.xsl, animateurs.xsl créent des extraits variés du programme.

## Fragments XSLT

```
<!-- Code qui fabrique la table des matières (jours) en HTML -->
Programme: <xsl:apply-templates select="//day" mode="toc" />

<xsl:template match="day" mode="toc">
  <a href="#{@name}{@dayno}{@month}"><xsl:value-of select="@name"/></a> -
</xsl:template>
<!-- Code pour insérer des attributs "name" dans le HTML -->
<xsl:template match="day">
   <a name="{@name}{@dayno}{@month}"><xsl:value-of select="@name"/></a> -
   <xsl:value-of select="@dayno"/>/<xsl:value-of select="@month"/>/
   <xsl:value-of select="@year"/>
</xsl:template>
```

## Fragment XML (élément à extraire)

- Faire une table des matières avec les jours de l'Atelier (avec liens)

```
<day name="lundi" year="2003" month="6" dayno="16">
```

## Résultat HTML

- Au début du fichier on a un menu qui affiche les jours (liens vers le bas)

```
Programme: <a href="#lundi166">lundi</a> - <a href="#mardi176">mardi</a> - ....
.....
```

- Dans le fichier on insère les attributs "name"

```
<a name="lundi166">lundi</a> - 16/6/2003
```

## Note:

- Dans ce répertoire il y a aussi un fichier programme-fo.xsl qui génère du code xsl-fo utilisé pour générer la version PDF du programme.

## Exemple 5-4: Tables de matières pour éléments sans ID

- Table de travaux pour une "page travaux STAF" (portfolio étudiant)

  *url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-toc/**

# 5.3 Numérotations

## Exemple 5-5: Numérotation d'éléments

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-numbering/**

Variante 1:

```
<xsl:template match="sections">
  <xsl:for-each select="section">
   <p>
    <xsl:number value="position()" format="a. "/>
    (<xsl:value-of select="deposit-date"/>) <a href="{url}"><xsl:value-of
select="title"/></a>
    .........
   </p>
  </xsl:for-each>
 </xsl:template>
```

Variante 2:

```
<xsl:template match="sections">
  <xsl:for-each select="section">
....
   <xsl:number level="multiple"
               count="exercise|section"
               format="1.1 "/>
....
```

# 6. Exécution de templates et appels

- XSLT est un véritable langage de programmation
- Ce chapitre s'addresse à des personnes ayant qqs. connaissances en informatique (programmation) !
- Toutefois, c'est un langage fonctionnel et qui demande une adaptation intellectuelle pour ceux qui savent par exemple programmer en PHP.

## 6.1 Le template (règle modèle)

- Les templates forment le coeur d'un programme XSLT
- Un template produit normalement un output et/ou fait appel à d'autres templates.

**Syntaxe:**

```
<!--Catégorie : élément de haut niveau -->
<xsl:template
      match = "expression XPath"
      name = "Nom"
      priority = "nombre"
      mode = "nom">
      <!-- Contenu : (xsl:param*, template)-->
</xsl:template>
```

### *Attribut match*

- Cet attribut déjà introduit définit un chemin de location et permet un déclenchement du template par "node matching".

### *Attribut name*

- Cet attribut permet de donner un nom au template. Par la suite une expression xsl:call-template peut explicitement appeler ce template comme fonction.

### *Attribut priority*

- Permet d'indiquer une priorité de traitement. Lorsque 2 templates correspondent à un noeud, le template qui a la priorité la plus haute sera choisi.
- Lorsque la prioté n'est pas décidé, les plus simples ont priorité. Voir la spécification pour plus de détails !

### *Attribut mode*

- Si un élément xsl:apply-templates a un attribut mode, alors il s'applique uniquement aux templates des éléments xsl:template ayant un attribut mode avec la même valeur.
- On utilise cette technique par exemple pour traiter 2 fois un noeud, la première fois par exemple pour créer une table des matières.
- Voir 5.2 "Fabrication de références (liens)" [19]

### *Les paramètres xsl:param*

- Lorsqu'on appelle un template, on peut lui passer des paramètres
- Un paramètre peut avoir une valeur par défaut

### *Le template*

- Le corps: Actions déclenchées par le template lorsqu'il est instancié (exécuté)

# 6.2 Variables et paramètres

- XSLT ne connait pas de variables au sens des langages procéduraux

## Les paramètres

### *Syntaxe*

```
<!-- Category: top-level-element -->
<xsl:param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:param>
```

- Un paramètre lie un nom à une valeur. La valeur est définie soit dans l'attribut select, soit par son contenu (mais pas les deux !)
- Lorsqu'on définit un paramètre à la racine, il s'applique à tous les templates qui font appel à ce paramètre et qui ne reçoivent pas sa valeur. Cela ressemble à une constante globale.
  - Exemple

```
<!-- space est un paramètre global -->
 <xsl:param name="space" select="10"/>
 <xsl:template name="render-course">
  <xsl:param name="position-y"/>
  <text x="{$space}" y="{$position-y * $increase-y - $space }"
        style="stroke:#000099;fill:#000099;font-size:12;">
   <xsl:value-of select="title"/>
  </text>
```

# Les variables

- Le nom "variable" n'est pas clair. Il s'agit en fait de constantes.
- En règle générale, c'est utile pour faire des calculs "intermédiaires"

### *Syntaxe*

```
<!-- Category: instruction -->
<xsl:variable
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:variable>
```

### Utilisation de paramètres et variables

**Syntaxe: $XYZ**

```
<xsl:param name="initial_size" select="5"/>
<xsl:param name="multiplier" select="3"/>

<xsl:variable name="fonte" select="$initial_size + $position * $multiplier"/>
<p style="font-size: {$fonte}pt ;"> blabla </p>
```

- Cf. Exemple 6-2: "XSLT avec des fonctions 2" [30] !

# 6.3 Appel d'un template (règle modèle)

### *Syntaxe*

```
<!--Catégorie : instruction -->
<xsl:call-template
  name = qname>
  <!-- Contenu : xsl:with-param*-->
</xsl:call-template>
```

### Exemple 6-1: XSLT avec des fonctions

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-call-template/ (répertoire)**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-call-template/news.xml**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-call-template/news-call-template.xsl**

- But: Afficher les title des item d'un contenu de type RSS de plus en grand

### Un fragment du XML

```
<news>
 <item>
   <title>Home Page de Daniel Schneider</title>
   <link>http://tecfa.unige.ch/tecfa-people/schneider.html</link>
    <description>Cette page renvoie à mes publications, ....</description>
  </item>
  <item>
   <title>Matériaux de cours de Daniel Schneider</title>
   <link>http://tecfa.unige.ch/guides/tie/tie.html</link>
   <description>Il s'agit de plusieurs centaines de transparents ... </description>
  </item>
  ....  </news>
```

## La feuille de style

```
<xsl:output method="html"
    encoding="ISO-8859-1"
    doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"/>

 <xsl:template match="/">
  <html>
   <head><title>Demo call-templates</title></head>
   <body bgcolor="#FFFFFF">
    <h1>Demo call-templates</h1>
    <xsl:apply-templates/>
   </body>
  </html>
 </xsl:template>
```

- Ce template appelé par le mécanisme d'application de règles contient une boucle qui, pour chaque enfant "item", fait explicitement appel au template "display_item"
- Lors de cet appel on transmet
  - le contexte (donc **"item"** courant)
  - plus un paramètre "position" qui prend la valeur de la position du "item" en question

```
<xsl:template match="news">
 <xsl:for-each select="item">
  <xsl:call-template name="display_item">
   <xsl:with-param name="position" select="position()"/>
  </xsl:call-template>
 </xsl:for-each>
</xsl:template>
```

- Ce template appelé (par call-template) reçoit un paramètre "position".
- On utilisera ce paramètre pour définir la taille de la police (font) pour le titre
- Donc:
  - **<xsl:param name="position"/>** définit un paramètre que ce template peut recevoir lorsqu'on l'appelle
  - **$position** utilise ce paramètre

```
<xsl:template name="display_item">
  <xsl:param name="position"/>
  <p style="font-size: {5 + $position * 5}pt ;">
    <a href="{link}"> <xsl:value-of select="title"/></a></p>
  <p><xsl:value-of select="description"/></p>
 </xsl:template>

</xsl:stylesheet>
```

# Exemple 6-2: XSLT avec des fonctions 2

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-call-template/ (répertoire)**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-call-template/news2.xml**

*url:* **http://tecfa.unige.ch/guides/xml/examples/xsl-call-template/news-call-template2.xsl**

- Cet exemple ressemble au précédant
- On ajoute:
  - Deux paramètres globaux qui règlent l'incrémentation de la police (font)
  - une variable **`fonte`** pour un calcul intermédiaire

```
<!-- you can change the value of these 2 parameters -->
 <xsl:param name="initial_size" select="5"/>
 <xsl:param name="multiplier" select="3"/>

 <xsl:template name="display_item">
  <xsl:param name="position"/>
  <xsl:variable name="fonte" select="$initial_size + $position * $multiplier"/>
  <p style="font-size: {$fonte}pt ;"><a href="{link}"><xsl:value-of select="title"/></
a><br />
   [Pos = <xsl:value-of select="$position"/>, Multip. = <xsl:value-of
select="$multiplier"/>,
   Taille init. = <xsl:value-of select="$initial_size"/>, Taille fonte = <xsl:value-of
select="$fonte"/>]
  </p>
  <p><xsl:value-of select="description"/></p>
 </xsl:template>
```