Référence XPath/XSLT

Bernd Amann et Philippe Rigaux

13 mars 2009

Cette annexe est consacrée aux éléments et aux fonctions XSLT. Elle est conçue et présentée comme une référence, venant compléter les chapitres « explicatifs » qui précèdent. Contrairement aux autres chapitres de ce livre qui suivaient un ordre basé sur la thématique et la mise en œuvre des fonctionnalités XSLT, la présentation suit l'ordre alphabétique pour faciliter la recherche. Nous donnons une description complète de chaque élément, en renvoyant si besoin est à la page du livre qui développe des explications et des exemples sur son utilisation.

Vous pouvez aussi vous aider de l'index pour trouver toutes les pages de ce livre qui traitent d'un élément XSLT donné, les pages relatives à la référence étant indiquées en **gras**.

1 Éléments XSLT

La liste des éléments XSLT a été donnée dans le chapitre **??**, avec une répartition en *éléments de premier niveau* (tableau **??** page **??**) et en *instructions* (tableau **??** page **??**). Dans ce qui suit nous reprenons, dans l'ordre alphabétique cette fois, tous ces éléments en donnant leur syntaxe complète, leur signification, et éventuellement quelques exemples complétant ceux que vous avez déjà pu trouver dans les chapitres qui précèdent.

Chaque élément est défini par son *type*, la liste de ses attributs, et le type de son contenu. Pour les attributs nous distinguons ceux qui sont obligatoires, indiqués en **`gras`**, de ceux qui sont optionnels, indiqués simplement avec une police `à chasse fixe`.

Il est possible d'exprimer des contraintes sur le type d'un attribut ou d'un élément sous la forme de *catégories syntaxiques* qui définissent la forme que peut prendre la valeur de l'attribut ou le contenu de l'élément. Quand c'est le cas nous l'indiquons dans la description de l'élément. Les catégories syntaxiques les plus courantes sont rappelées dans le tableau 1.

Nous utilisons les conventions des DTD pour indiquer éventuellement l'ordre ou la multiplicité d'apparition de telle ou telle catégorie syntaxique dans le contenu d'un élément : `(cat)`* indique une catégorie qui peut être répété 0 ou plusieurs fois, `(cat)?` indique une catégorie optionnelle, `(cat1 | cat2)`* donne le choix entre deux catégories, etc.

xsl:apply-imports

Syntaxe

`<xsl :apply-imports/>`

Description

Quand on importe un programme XSLT dans un autre, les règles du programme importé ont une préséance inférieure à celles du programme importateur (voir page **??**). Une des conséquences est que la règle importée est *remplacée*, ce qui n'est pas toujours l'effet souhaité. On peut très bien vouloir utiliser cette règle, et la *compléter* par d'autres instructions. Dans ce cas on utilise, dans le corps de la nouvelle règle, l'instruction `xsl:apply-imports`.

Désignation	**Description**
`Expression`	Toute expression XPath (voir chapitre **??**, page **??**).
`NameChar`	Les caractères acceptés dans un nom XML : lettres, chiffres, «`.`», «`_`», «`-`», et «`:`». On ajoute également les caractères de combinaison (*combining character*) et caractères d'extension (*extender*) qui correspondent à des caractères de langues étrangères avec des accents et diacritiques (voir la recommandation XML pour les codes Unicode de ces caractères).
`Espace`	Un espace est une séquence de caractères identifiés par les codes Unicode *x9* (tabulation), *xA* (saut de ligne), *xD* (retour chariot) et *x20* (blanc).
`Name`	La recommandation XML définit un *nom* comme une séquence de caractères qui commence par une lettre, «`_`» ou «`:`» suivie par une séquence de `NameChar`. La recommandation pour les espaces de noms raffine cette définition en distinguant les noms sans «`:`» (`NCName`) et les noms ayant au maximum un caractère «`:`» (`QName`).
`NCName`	Cette catégorie désigne tout nom XML (`Name`) valide et sans le caractère «`:`».
`QName`	Un *nom qualifié* est un `Name` avec au maximum une occurrence du caractère «`:`». Si présent, «`:`» sépare le nom en *prefixe* (`NCName`) et en *nom local* (`NCName`). Par exemple `xbook :monEl` est composé du préfixe `xbook` et du nom local `monEl`.
`QNames`	Une séquence de noms qualifiés (`QName`) séparés par des espaces (`Espace`).
`NMToken`	Toute séquence non-vide de caractères `NameChar` (sans espace blanc).
`CData`	Chaînes de caractères contenues dans les nœuds de type **Text**, sections CData ou des attributs de type `CDATA`. Essentiellement toutes les chaînes de caractères sans les caractères '<', '>' et '&'.
`Pattern`	Les *patterns* appartiennent au sous-ensemble des expressions XPath autorisées dans l'attribut `match` de l'élément `xsl:template` : voir page **??**.
`URL`	La définition des *Uniform Resource Locators* (`URL`) fait partie du protocole HTTP. Une URL prend le plus souvent la forme d'un chemin de fichier ou d'une adresse *`Protocole`*`//`*`:adrServeur/cheminLoc`* où *`Protocole`* désigne un protocole comme `http` ou `ftp`, *`adrServeur`* est l'adresse IP du serveur et *`cheminLoc`* est un chemin local sur ce serveur.
`URI`	Un *Uniform Resource Identifier* est une généralisation de `URL` vers d'autres systèmes d'identification standards comme les numéros ISBN. Chaque URL est une URI.
`corps de règle`	Un corps de règle (voir page **??**) est toute combinaison de texte littéral, d'éléments littéraux (c'est-à-dire non interprétés par le processeur) et d'instructions XSLT correctement formées. Un corps de règle définit un fragment du document XML à instancier au cours du traitement.
`CodeLangue`	Code d'une langue comme par exemple `fr` pour le Français, `de` pour l'Allemand ou `en-US` pour l'anglais écrit et parlé aux États-Unis.

TAB. 1 – Catégories syntaxiques

Cette fonctionnalité s'apparente au mécanisme de surcharge de méthode dans la programmation orientée-objet. Notez qu'il est souvent possible d'obtenir le même résultat avec `xsl:call-template` ou en utilisant l'attribut `mode` pour choisir parmi différentes règles possibles.

La version 2.0 de XSLT prévoit la possibilité de passer des paramètres à la règle importée en plaçant dans son contenu une ou plusieurs instructions `xsl:with-param` (voir page **37**).

Exemple

Voici par exemple un programme XSLT qui contient une règle de présentation standard pour le contenu d'un élément <PERSONNE> (nom, prénom, date de naissance) :

Exemple .1 *Personne.xsl* : *Une règle générique pour les personnes*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <head><title>Ficher personne</title></head>
      <body bgcolor="white">
        <xsl:apply-templates/>
      </body></html>
  </xsl:template>

  <xsl:template match="PERSONNE">
    <b> Prénom</b> : <xsl:value-of select="PRENOM"/>,
    <b> Nom </b>   :<xsl:value-of select="NOM"/>,
        Né le <i><xsl:value-of select="NAISSANCE"/></i>
  </xsl:template>

</xsl:stylesheet>
```

Maintenant supposons qu'on se trouve avec une occurrence d'un élément de type `PERSONNE` plus détaillée, comme celle du fichier ci-dessous.

Exemple .2 *JohnDoe.xml* : *Le salarié John Doe*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<PERSONNE NOSS="17517652987">
    <NOM>Doe</NOM>
    <PRENOM>John</PRENOM>
    <NAISSANCE>15:10:1981</NAISSANCE>
    <POSTE>Directeur Informatique</POSTE>
    <SALAIRE>50 000</SALAIRE>
</PERSONNE>
```

Au lieu de redéfinir complètement la règle du programme *Personne.xsl*, on peut *l'importer*, puis l'utiliser avec `xsl:apply-import` inséré dans la nouvelle règle de traitement des éléments de type `PERSONNE`

Exemple .3 *ApplyImports.xsl* : *Utilisation de* `xsl:apply-imports`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:import href="Personne.xsl"/>
  <xsl:output method="html" indent="yes"/>
```

```
  <xsl:template match="/">
    <html>
      <head><title>Ficher personne</title></head>
      <body bgcolor="white">
        <xsl:apply-templates/>
      </body></html>
  </xsl:template>

  <xsl:template match="PERSONNE">
     <xsl:apply-imports/>

    <b> Poste</b> : <xsl:value-of select="POSTE"/>,
    <b> Salaire </b>   :<xsl:value-of select="SALAIRE"/>
  </xsl:template>

</xsl:stylesheet>
```

Une règle est utilisée dans `xsl:apply-imports` si et seulement si elle s'applique au nœud courant et si elle a été importée. Une solution simple pour être sûr qu'une règle importée et une règle locale s'appliquent au même nœud est bien entendu qu'elles aient toutes deux le même *pattern*.

xsl:apply-templates

Syntaxe

```
<xsl :apply-templates
    select=Expression
    mode=QName>
        (<xsl :with-param> | <xsl-sort>)*
</xsl :apply-templates>
```

Description

Cette instruction désigne des nœuds à traiter au sein du document source (ou, éventuellement, d'un document référencé par la fonction *document*()). Le processeur détermine alors pour chaque nœud sélectionné la règle à appliquer. Les deux attributs possibles sont optionnels. L'attribut `select` est une expression XPath dont l'évaluation doit donner un ensemble de nœuds. Si cette expression est relative, le nœud contexte est le nœud du document source qui a instancié la règle contenant `xsl:apply-templates` (ou nœud courant : voir page **??**). Si `select` est absent, alors les nœuds sélectionnés sont les fils du nœud courant (autrement la valeur par défaut de `select` est `child : :node()`).

L'attribut `mode` indique un critère complémentaire pour la sélection des règles à appliquer. Si cet attribut est présent avec une valeur *`nomMode`*, seules les règles `xsl:template` ayant elles-mêmes un attribut `mode` égale à *`nomMode`* seront prises en compte (voir page **??**).

On peut passer des paramètres aux règles déclenchées importée en plaçant dans le contenu de `xsl:apply-templates` une ou plusieurs instructions `xsl:with-param` (voir page **37**).

Par défaut les éléments sélectionnés sont pris en compte dans l'ordre de parcours du document. En introduisant un ou plusieurs éléments `xsl:sort`, on peut changer cet ordre : voir page **30**.

Exemples

`xsl:apply-templates` est bien entendu l'un des éléments les plus utilisés dans un programme XSLT. De très nombreux exemples ont été donnés dans ce livre : voir notamment la section consacrée au règles, page **??** du chapitre **??**.

Syntaxe

```
<xsl :attribute
    name=QName
    namespace=URI>
        corps de règle
</xsl :attribute>
```

Description

Cet élément déclenche l'ajout d'un attribut dans l'élément courant du document résultat. Cet élément courant peut être soit un élément littéral, soit un élément créé avec l'instruction `xsl:element`, soit un élément du document source copié avec `xsl:copy`. L'élément `xsl:attribute` doit être instancié immédiatement après la balise ouvrante de l'élément courant.

L'attribut `name` est obligatoire et désigne le nom de l'attribut à créer. L'attribut optionnel `namespace` permet de définir un espace de noms pour l'attribut. Dans le cas où le nom de l'attribut contient déjà le préfixe d'un autre espace de noms, celui sera remplacé par l'espace de noms `namespace` (voir chapitre **??**).

Le contenu de `xsl:attribute` définit la valeur de l'attribut. C'est un corps de règle qui peut contenir d'autres instructions XSLT, et qui doit impérativement engendrer un nœud de type **Text** (donc sans marquage).

Il est possible de *calculer* les deux attributs, `name` et `namespace`, en utilisant des expressions XPath encadrées par { } (*attribute value template*). Ce même mécanisme est applicable pour placer directement les attributs dans l'élément, sans avoir donc besoin d'utiliser `xsl:attribute`. Voici quelques exemples pour illustrer ces possibilités (voir également page **??**).

Exemples

Reprenons le document *JohnDoe.xml*, page 3. Il contient un élément racine de type `PERSONNE`, avec des fils `NOM` et `PRENOM`. Le programme suivant crée un document contenant un seul élément, les fils étant transformés en attributs.

Exemple .4 *Attribute1.xsl* : *L'élément* `xsl:attributes`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="PERSONNE">
    <PERSONNE>
     <xsl:attribute name="NOM">
           <xsl:value-of select="NOM"/>
     </xsl:attribute>
     <xsl:attribute name="PRENOM">
           <xsl:value-of select="PRENOM"/>
     </xsl:attribute>
    </PERSONNE>
  </xsl:template>

</xsl:stylesheet>
```

Le résultat de la transformation est :

```
<?xml version="1.0" encoding="UTF-8"?>
<PERSONNE NOM="Doe" PRENOM="John"/>
```

Maintenant on obtient exactement le même résultat avec le programme suivant :

Exemple .5 *Attribute2.xsl* : *Un programme équivalent, sans* `xsl:attributes`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="PERSONNE">
    <PERSONNE NOM="{NOM}" PRENOM="{PRENOM}"/>
  </xsl:template>

</xsl:stylesheet>
```

Alors quel est l'intérêt d'utiliser `xsl:attribute` ? Et bien dans tous les cas où on ne veut pas l'associer à un élément littéral (par exemple quand l'élément est engendré par `xsl:element`), ou quand le nom ou la valeur de l'attribut sont calculés par des expressions complexes.

Voici un exemple où on veut ajouter un attribut AGE à l'élément <PERSONNE> dans le document résultat, en calculant la valeur de cet attribut par différence entre l'année courante (disons que c'est 2002) et l'année de naissance :

Exemple .6 *Attribute3.xsl* : *L'élément* `xsl:attributes`*, avec une valeur calculée*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="PERSONNE">
   <PERSONNE NOM="{NOM}" PRENOM="{PRENOM}">
   <xsl:attribute name="AGE">
     <xsl:choose>
       <xsl:when test="NAISSANCE/ANNEE">
         <xsl:value-of select="2002 - NAISSANCE/ANNEE"/>
       </xsl:when>
       <xsl:when test="NAISSANCE/DATE">
         <xsl:value-of select="2002 - substring(NAISSANCE/DATE,7)"/>
       </xsl:when>
     </xsl:choose>
   </xsl:attribute>
 </PERSONNE>
</xsl:template>

</xsl:stylesheet>
```

La règle distingue entre deux possibilités : soit l'année est stockée explicitement comme élément de type ANNEE ou elle doit être extraite avec la fonction *substring*() d'une chaîne de caractères DATE contenant également le jour et le mois de naissance.

xsl:attribute-set

Syntaxe

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

Description

Cet élément permet de grouper des définitions d'attributs, et de nommer les groupes ainsi constitués afin de pouvoir les affecter d'un bloc à un élément. Cette fonctionnalité est assez proche des feuilles de style (CSS) dans lesquelles on factorise des propriétés de mise en forme de certains éléments HTML.

L'attribut **`name`** est obligatoire : il sert à référencer le groupe d'attributs. Le second attribut, `use-attribute-set`, permet de composer récursivement un groupe d'attributs à l'aide d'autres groupes. La valeur de `use-attribute-set` doit être une liste de noms de groupe d'attributs, séparés par des espaces. On peut ainsi par exemple définir un groupe pour les attributs relatifs aux polices de caractères, un autre pour les couleurs, et les grouper tous deux dans un troisième groupe.

Il est possible de trouver plusieurs groupes d'attributs avec le nom dans un programme XSLT. Dans ce cas le processeur fusionne les deux listes.

Pour affecter un groupe d'attributs à un élément, on peut utiliser l'attribut `use-attribute-sets` des éléments `xsl:copy` ou `xsl:element` (voir pages **11** et **12**), ou introduire un attribut `xsl :use-attribute-sets` dans un élément littéral.

Exemple

Le programme suivant définit un groupe nommé `MonStyle` avec des attributs de présentation reconnus par HTML. Ce groupe est ensuite appliqué à l'élément <`body`>.

Exemple .7 *AttributeSet.xsl* : *L'élément* `xsl:attribute-set`

```
<?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>
```

L'application de ce programme produit le résultat suivant :

Exemple .8 *AttributeSet.html* : *Résultat du programme précédent*

```
<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>
```

xsl:call-template

Syntaxe

```
<xsl :call-template name=QName>
        (<xsl :with-param>)*
</xsl :call-template>
```

Description

L'élément `xsl:call-template` permet d'appeler une règle par son nom. Il est utile dès qu'un fragment doit être instancié à plusieurs endroits d'un programme. Pour éviter des répétitions, on peut placer ce fragment dans une règle, et appeler cette règle avec `xsl:call-template`. Le ou les éléments `xsl:with-param` peuvent être utilisés pour passer des paramètres à la règle : voir page **37**.

La fonctionnalité apportée par cet élément s'apparente à un appel de fonction dans un langage de programmation classique. Il faut noter cependant qu'il n'y a pas de valeur retournée, ni de modification des « arguments » définis par `xsl:with-param`. Une des manières de contourner ces restrictions est d'inclure l'appel à `xsl:call-template` dans un élément `xsl:variable` : voir page **36**.

Exemples

Voir la section consacrée aux règles nommées, page **??**, pour des exemples.

xsl:choose

Syntaxe

```
<xsl :choose>
        (<xsl :when>)+
        (<xsl :otherwise>)?
</xsl :choose>
```

Description

Cet élément est associé à `xsl:when` et `xsl:otherwise` pour créer l'équivalent des structures de test que l'on trouve habituellement dans les langages de programmation (`if-then-else` ou `switch`). Son contenu est une liste d'éléments `xsl:when` (au moins un), chacun étant associé à un test, et chacun ayant un contenu sous forme de corps de règle. Le premier élément `xsl:when` pour lequel le test s'évalue à `true` voit son corps de règle instancié. Les éléments qui suivent sont alors ignorés. Si aucun `xsl:when` ne s'évalue à `true`, le contenu de l'élément `xsl:otherwise`, s'il existe, est instancié.

Exemples

Voici un document représentant (très synthétiquement) le présent livre.

Exemple .9 *XBook.xml* : *Une description du livre*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<XBOOK ISBN="1-67098-09">
    <TITRE>Publication web avec XML/XSLT</TITRE>
    <EDITEUR>Editions O'Reilly France</EDITEUR>
    <AUTEURS>
      <AUTEUR><NOM>Bernd Amann</NOM>
              <AFFILIATION>Cnam</AFFILIATION>
      </AUTEUR>
```

```
      <AUTEUR><NOM>Philippe Rigaux</NOM>
              <AFFILIATION>Université Paris-Sud</AFFILIATION>
      </AUTEUR>
     </AUTEURS>
     <CHAPITRE LONGUEUR="8">
       <TITRE>Avant-propos</TITRE>
     </CHAPITRE>
     <CHAPITRE LONGUEUR="43">
       <TITRE>Introduction à XML/XSLT</TITRE>
     </CHAPITRE>
     <CHAPITRE LONGUEUR="34">
       <TITRE>Documents XML : Structure et navigation</TITRE>
       <MOT-CLE>DOM</MOT-CLE><MOT-CLE>XPath</MOT-CLE>
     </CHAPITRE>
     <CHAPITRE LONGUEUR="45">
       <TITRE>Programmation XSLT</TITRE>
       <MOT-CLE>XSLT</MOT-CLE>
     </CHAPITRE>
</XBOOK>
```

Le programme suivant affiche tous les titres des chapitres du livre en les classant en trois catégories : chapitres « courts », « moyens » et « longs ». Le classement s'effectue en fonction de la valeur de l'attribut `LONGUEUR` de chaque chapitre.

Exemple .10 *RefChoose.xsl* : *Exemple d'utilisation de* `xsl:choose`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="XBOOK">
    <CHAPITRES>
      <xsl:apply-templates select="CHAPITRE"/>
    </CHAPITRES>
  </xsl:template>

  <xsI:template match="CHAPITRE">
   <xsl:choose>
     <xsl:when test="@LONGUEUR &lt; 20">
       Le chapitre "<xsl:value-of select="TITRE"/>" est court
      </xsl:when>

     <xsl:when test="@LONGUEUR &lt; 30">
       La longueur du chapitre "<xsl:value-of select="TITRE"/>" est moyen
      </xsl:when>

     <xsl:when test="@LONGUEUR &gt;= 30">
       Le chapitre "<xsl:value-of select="TITRE"/>" est long
      </xsl:when>

      <xsl:otherwise>
       Le chapitre "<xsl:value-of select="TITRE"/>"
          n'est ni court, ni moyen, ni long !
      </xsl:otherwise>
   </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

L'attribut `test` a pour valeur une expression XPath quelconque qui est convertie en booléen selon les règles présentées dans le chapitre **??**, page **??**. Noter que l'opérateur « < » ne peut être utilisé directement dans le test et doit être remplacé par une référence à l'entité prédéfinie `<`.

Exemples

Voir également page **??**.

xsl:comment

Syntaxe

```
<xsl :comment>
    corps de règle
</xsl :comment>
```

Description

Cette instruction permet d'inclure un commentaire dans le document résultat. Bien entendu il est illusoire de placer un commentaire sous la forme habituelle `<!-- ...-->` dans le programme XSLT en espérant le copier dans le document résultat, puisqu'il sera tout simplement ignoré par le processeur. Le contenu de l'élément peut être un corps de règle avec des instructions XSLT, du texte, des éléments littéraux, etc. Il faut éviter d'y introduire deux tirets `--`. L'instanciation du corps de règle sera introduit dans le document résultat sous la forme :
`<!-- instanciation du corps de règle -->`

L'introduction de commentaires peut être utile pour comprendre, en lisant le document résultat, quelles sont les règles qui ont été instanciées et d'où proviennent les informations.

Exemples

Le programme suivant produit la liste de tous les mots-clés présents dans *XBook.xml*, avec un commentaire indiquant de quels chapitres proviennent les mots-clés.

Exemple .11 *Comment.xsl* : *Exemple d'utilisation de* `xsl:comment`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="XBOOK">
    <MOTS-CLES>
      <xsl:apply-templates select="CHAPITRE/MOT-CLE"/>
    </MOTS-CLES>
  </xsl:template>

  <xsl:template match="MOT-CLE">
    <xsl:comment>
      Ces mots-clés sont issus du chapitre
         <xsl:value-of select="../TITRE"/>
    </xsl:comment>
    <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

xsl:copy

Syntaxe

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

Description

Cette instruction copie le nœud courant dans le document résultat. Cette copie comprend le balisage et la valeur, mais n'effectue pas de copie en profondeur. Les éléments sont donc copiés sans leur contenu.

L'attribut `use-attribute-set` permet d'introduire des attributs dans l'élément copié en faisant référence au nom d'un groupe d'attributs : voir page **6**.

On peut introduire un contenu dans un élément copié en plaçant un corps de règle dans `xsl:copy`. Voir les exemples donnés page **??**.

xsl:copy-of

Syntaxe

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

Description

L'élément `xsl:copy-of` déclenche une copie *en profondeur* de tous les nœuds sélectionnés par l'expression de l'attribut `select`. La copie en profondeur implique que tous les descendants sont copiés également. Cela recouvre notamment, pour les éléments, la copie de tous les éléments descendants, avec leurs attributs (qui sont *aussi* des nœuds) et leur contenu textuel.

Exemples

Voir les exemples donnés page **??**.

xsl:decimal-format

Syntaxe

```
<xsl :decimal-format
    name=QName
    NaN=CData
    infinity=CData
    pattern-separator=caractère
    decimal-separator=caractère
    grouping-separator=caractère
    minus-sign=caractère
    percent=caractère
    per-mille=caractère
    zero-digit=caractère
    digit=caractère />
```

Description

Il s'agit d'un élément du premier niveau qui est utilisé en association avec la fonction *format-number*() pour transformer un entier en chaîne de caractères. Tous les attributs sont optionnels. L'attribut `name` permet de référencer un format dans l'appel de la fonction *format-number*(). En absence de cet attribut `xsl:decimal-format` définit le format par défaut.

La signification et les valeurs par défaut pour les autres attributs sont donnés dans le tableau 2.

Attribut	Signification	Défaut
`infinity`	chaîne de caractères qui représente la valeur infinie (division par 0)	`'Infinity'`
`NaN`	chaîne de caractères qui représente une valeur qui n'est pas un nombre	`'NaN'`
`pattern-separator`	caractère séparant le format des nombres positifs de celui des nombres négatifs	`;`
`decimal-separator`	séparateur entre la partie entière et la fraction d'un nombre	`.`
`grouping-separator`	séparateur de groupes de chiffres	`,`
`minus-sign`	caractère utilisé pour distinguer les nombres négatifs	`-`
`percent`	caractère pourcent	%
`per-mille`	caractère pourmille	‰
`zero-digit`	caractère indiquant l'emplacement d'un chiffre. Les chiffres manquants sont remplacés par ce caractère	`0`
`digit`	caractère indiquant l'emplacement d'un chiffre. Les chiffres manquants ne sont pas remplacés.	#

TAB. 2 – Signification et valeurs par défaut des attributs de l'élément `xsl:decimal-format`

Exemples

Voir la fonction *format-number*(), page **39**.

xsl:element

Syntaxe

```
<xsl :element
    name=QName
    namespace=URI
    use-attribute-sets=QNames>
        corps de règle
</xsl :element>
```

Description

Cette instruction permet d'insérer un élément dans le document résultat. Son principal intérêt est de permettre de déterminer dynamiquement (au moment de l'exécution du programme) le nom ou l'espace de noms de l'élément créé, tous les autres cas pouvant se ramener à l'insertion d'un élément littéral. Voir page **??** une description de cette instruction.

L'attribut `use-attribute-sets` permet de faire référence à un groupe d'attributs nommé par `xsl:attribute-set` : voir page **6**.

Exemples

Le programme suivant transforme tous les attributs de l'élément racine du document *XBook.xml* (voir page 8) en éléments, et copie en profondeur les éléments <TITRE> et <EDITEUR>.

Exemple .12 *Element.xsl* : *Exemple d'utilisation de* `xsl:element`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="ISO-8859-1"/>

  <xsl:template match="/XBOOK">
    <LIVRE>
      <xsl:copy-of select="TITRE"/>

      <xsl:for-each select="@*">
        <xsl:element name="{name()}">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>

      <xsl:copy-of select="EDITEUR"/>
    </LIVRE>
  </xsl:template>
</xsl:stylesheet>
```

xsl:fallback

Syntaxe

```
<xsl :fallback>
        corps de règle
</xsl :fallback>
```

Description

Cette instruction s'exécute quand le père de l'élément n'est pas reconnu par le processeur XSLT. Il est donc principalement destiné à permettre des extensions de XSLT tout en préservant dans une certaine mesure la portabilité des programmes.

Exemples

Voici une version améliorée du programme *SiteEquipe.xsl* qui prend en compte une éventuelle absence de l'instruction `xsl:document` en la remplaçant par l'instruction `xalan :write` fournie par Xalan.

Exemple .13 *Fallback.xsl* : *Illustration de* `xsl:fallback`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xalan="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="xalan" >

<xsl:template match="EQUIPE">

  <xsl:document href="EQUIPE{@NUMERO}/index.html">
```

```
    <xsl:call-template name="genEquipe"/>
    <xsl:fallback>
      <xalan:write file="EQUIPE{@NUMERO}/index.html" >
        <xsl:call-template name="genEquipe"/>
      </xalan:write>
    </xsl:fallback>
  </xsl:document>

</xsl:template>

<xsl:template match="MEMBRE">

  <xsl:document href="{@NOM}.html">
    <xsl:call-template name="genMembre"/>
    <xsl:fallback>
      <xalan:write file="{@NOM}.html" >
        <xsl:call-template name="genMembre"/>
      </xalan:write>
    </xsl:fallback>
  </xsl:document>

</xsl:template>

<xsl:template name="genEquipe">
  <html>
    <!-- page de l'equipe -->
  </html>
  <xsl:apply-templates select="MEMBRE"/>
</xsl:template>

<xsl:template name="genMembre"
  <html>
    <!-- page de membre -->
  </html>
</xsl:template>

</xsl:stylesheet>
```

xsl:for-each

Syntaxe

```
<xsl :for-each
    select=Expression>
        (<xsl :sort> *, corps de règle)
</xsl :for-each>
```

Description

L'instruction `xsl:for-each` instancie son contenu pour chacun des nœuds désignés par l'expression de l'attribut `select`. Le nœud est pris à chaque fois comme nœud courant, et le contexte pour l'ensemble de la boucle est constitué de l'ensemble des nœuds sélectionnés par le `select`. L'instruction `xsl:for-each` change donc temporairement le contexte et le nœud courant d'une règle.

Exemples

Voir la description et les cas d'utilisation de cette instruction dans le chapitre **??**, page **??**.

xsl:if

Syntaxe

```
<xsl :if
    test=Expression>
        corps de règle
</xsl :if>
```

Description

Cette instruction instancie son contenu si l'expression de l'attribut `test` s'évalue à `true`. Il n'y a pas d'équivalent en XSLT au `else` habituellement trouvé dans la syntaxe des langages de programmation, mais on peut utiliser l'instruction `xsl:choose` qui est beaucoup plus puissante : voir page **8**.

Exemples

Voir la description et les cas d'utilisation de cette instruction dans le chapitre **??**, page **??**.

xsl:import

Syntaxe

```
<xsl :import href=URI/>
```

Description

Cet élément de premier niveau importe un autre programme XSLT avec tous ses éléments de premier niveau et toutes ses règles. La précédence de tous les éléments importés est inférieure à ceux du document principal.

Cet élément doit apparaître avant tous les autres éléments de premier niveau parmi les enfants de l'élément racine `xsl:stylesheet`. Nous renvoyons au chapitre **??**, page **??** pour une description détaillée des règles d'importation de documents.

xsl:include

Syntaxe

```
<xsl :include href=URI/>
```

Description

`xsl:include` insère dans le programme le contenu du document XSLT référencé par l'attribut `href`. Tout se passe ensuite comme si les éléments importés avaient été présent dans le programme principal dès l'origine. Contrairement à `xsl:import`, il n'existe donc pas d'ordre de préséance entre les éléments inclus et les autres.

xsl:key

Syntaxe

```
<xsl :key
    name=QName
    match=Pattern
    use=Expression/>
```

Description

XSLT permet de définir des groupes de nœuds dans le document source (attribut `match`), de référencer ces groupes par un nom (attribut `name`), et enfin d'indiquer une expression qui calcule pour chaque nœud du groupe une clé (attribut `use`). Tous ces attributs sont obligatoires.

Un fois un groupe (une « clé » dans la terminologie XSLT) défini, on peut extraire un nœud avec la fonction *key*(). La combinaison de l'élément `xsl:key` et de la fonction *key*() permet de gérer des liens dans les documents résultats, et notamment dans les sites HTML.

Exemples

Voici un document XML contenant quelques films avec leurs metteurs en scène.

Exemple .14 *Films.xml* : *Quelques films*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<FILMS>
<FILM ANNEE="1979">
<TITRE>Alien</TITRE><AUTEUR>Scott</AUTEUR>
</FILM>
<FILM ANNEE="1958">
<TITRE>Vertigo</TITRE><AUTEUR>Hitchcock</AUTEUR>
</FILM>
<FILM ANNEE="1960">
<TITRE>Psychose</TITRE><AUTEUR>Hitchcock</AUTEUR>
</FILM>
<FILM ANNEE="1980">
<TITRE>Kagemusha</TITRE><AUTEUR>Kurosawa</AUTEUR>
</FILM>
<FILM ANNEE="1997">
<TITRE>Volte-Face</TITRE><AUTEUR>Woo</AUTEUR>
</FILM>
<FILM ANNEE="1997">
<TITRE>Titanic</TITRE><AUTEUR>Cameron</AUTEUR>
</FILM>
<FILM ANNEE="1986">
<TITRE>Sacrifice</TITRE><AUTEUR>Tarkovski</AUTEUR>
</FILM>
</FILMS>
```

Le programme XSLT suivant définit un groupe de nom `FilmsDate` contenant tous les films indexés par l'attribut `ANNEE` de l'élément <`FILM`>.

Exemple .15 *Key.xsl* : *Utilisation de* `xsl:key`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:key name="FilmsDate" match="//FILM" use="@ANNEE"/>

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select='key("FilmsDate", "1997")'>
      <xsl:value-of select="TITRE"/><xsl:text> </xsl:text>
    </xsl:for-each>
  </xsl:template>
```

```
</xsl:stylesheet>
```

On peut alors utiliser la fonction *key*() pour désigner un ou plusieurs des éléments du groupe en fonction d'une valeur de la clé. Le programme ci-dessus désigne par exemple tous les films de 1997 (soit *Volte-Face* et *Titanic*). On peut noter que *key*() s'intègre à des expressions XPath complexes puisque son évaluation donne un ensemble de nœuds à partir desquels on peut exprimer d'autres étapes XPath. Voir page **??** pour une description plus complète et des exemples.

xsl:message

Syntaxe

```
<xsl :message
    terminate=("yes" | "no")>
        corps de règle
</xsl :message>
```

Description

Cette instruction permet d'afficher un message sur la « console » du processeur XSLT. Elle peut principalement être utilisée pour suivre le déroulement de l'exécution du programme. L'attribut `terminate` avec la valeur `yes` permet de forcer l'interruption du programme.

xsl:namespace-alias

Syntaxe

```
<xsl :namespace-alias
    stylesheet-prefix=préfixe | "#default"
    result-prefix=préfixe | "#default" />
```

Description

Cet élément demande au processeur XSLT de modifier le préfixe de certains éléments littéraux quand ils sont recopiés dans le document résultat. Le préfixe présent dans le programme est indiqué par `stylesheet-prefix`, et son équivalent dans le document résultat est donné par `result-prefix`. Les deux attributs sont obligatoires. Cette instruction est surtout importante pour créer des programmes XSLT.

Exemples

Supposons que le but d'un programme XSLT soit de produire un autre programme XSLT. On trouverait alors typiquement des règles comme celles du programme ci-dessous, qui sont évidemment invalides puisque la présence d'un `xsl:stylesheet` n'est pas autorisée dans un corps de règle.

Exemple .16 *NamespaceAlias1.xsl* : *Problème d'un élément littéral avec un préfixe* `xsl :`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
      <xsl:stylesheet version="1.0">
        <xsl:apply-templates />
      </xsl:stylesheet>
```

```
  </xsl:template>

</xsl:stylesheet>
```

La solution consiste à utiliser, dans le programme, un préfixe spécial, par exemple `myxsl`, pour les éléments littéraux, et à convertir ce préfixe spécial en `xsl` au moment de l'inclusion de ces éléments littéraux dans le document résultat. La conversion est indiquée avec `xsl:namespace-alias`, comme dans l'exemple ci-dessous.

Exemple .17 *NamespaceAlias2.xsl* : *Utilisation de* `xsl:NamespaceAlias`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:namespace-alias
         stylesheet-prefix="myxsl"
         result-prefix="xsl"/>

  <xsl:template match="/">
      <myxsl:stylesheet version="1.0">
        <xsl:apply-templates/>
      </myxsl:stylesheet>
  </xsl:template>

</xsl:stylesheet>
```

Voir un autre exemple dans le chapitre **??** page **??**.

xsl:number

Syntaxe

```
<xsl :number
    level="single" | "multiple" | "any"
    count=Pattern
    from=Pattern
    value=Expression
    format=CData
    lang=CodeLangue
    letter-value="alphabetic" | "traditional"
    grouping-separator=caractère
    grouping-size=entier />
```

Description

Cet élément est utilisé pour numéroter les nœuds du document source. Cela est notamment utile pour les documents structurés en chapitres/sections/sous-sections, comme nous l'avons vu page **??**.

L'élément produit soit un nombre, typiquement la position d'un nœud dans un ensemble, soit une combinaison de nombres représentant la position dans un arbre. Ce nombre (ou cette combinaison) peut être mis en forme de manières très diverses avec l'attribut `format`.

Nous utiliserons l'exemple suivant pour montrer l'utilisation de `xsl:number`. Il s'agit d'un extrait du découpage d'un enregistrement consacré à la Passion selon Saint Mathieu de Johann-Sebastian Bach. Ce découpage s'effectue d'abord en plusieurs CDs, puis en plusieurs plages sur chaque CD, enfin chaque plage contient une ou plusieurs séquences de l'œuvre.

Exemple .18 *Matthaus.xml* : *Un enregistrement de la Passion selon St Matthieu*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<DISQUE>
  <OEUVRE>Die Matthäus-Passion</OEUVRE>
  <AUTEUR>Johann-Sebastian Bach</AUTEUR>
  <CD>
    <PLAGE>
      <S>Chorus: Kommt, ihr Töchter, helft mir klagen</S>
    </PLAGE>
    <PLAGE>
      <S>(Evangelist, Jesus): Da Jesus diese Rede vollendet
         hatte</S>
    </PLAGE>
    <PLAGE>
      <S>Choral: Herzliebster Jesu, was hast du verbrochen</S>
    </PLAGE>
    <PLAGE>
      <S>(Evangelist): Da versammelten sich die Hohenpriester</S>
      <S>(Chori): Ja nicht auf das Fest</S>
      <S>(Evangelista): Da nun Jesus war zu Bethanien</S>
      <S>(Chorus): Wozu dienet dieser Unrat</S>
      <S>(Evangelista, Jesus): Da das Jesus merkte</S>
   </PLAGE>
 </CD>
 <CD>
   <PLAGE>
     <S>(Evangelist, Jesus, Judas):  Und er kam und fand sie
        aber schlafend</S>
   </PLAGE>
   <PLAGE>
     <S>Aria: So ist mein Jesus nun gefangen</S>
     <S>(Chori): Sind Blitze, sind Donner in Wolken verschwunden</S>
   </PLAGE>
  </CD>
</DISQUE>
```

Numérotation simple

La première utilisation de `xsl:number` consiste à numéroter les nœuds dans un ensemble par leur position.

Exemple .19 *Number1.xsl* : *Première application de* `xsl:number`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"
         encoding="ISO-8859-1"/>

  <xsl:variable name="sautLigne"><xsl:text>
  </xsl:text></xsl:variable>

  <xsl:variable name="espace">
    <xsl:text> </xsl:text>
  </xsl:variable>

  <xsl:template match="DISQUE">
    <xsl:value-of select="concat(OEUVRE, $sautLigne, AUTEUR, $sautLigne)"/>
```

```
      <xsl:for-each select="CD/PLAGE/S">
        <xsl:number value="position()" format="1. "/>
        <xsl:value-of select="concat(., $sautLigne)"/>
    </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

Dans ce premier exemple on utilise `xsl:number` avec les attributs `format` et `value`. La valeur de l'attribut `value` est obtenue par la fonction *position*(), et le format défini par l'attribut `format` est un nombre suivi d'un point et d'un espace. Le résultat est montré dans le document *Number1.txt*.

Exemple .20 *Number1.txt* : *Résultat du programme précédent*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  1. Chorus: Kommt, ihr Töchter, helft mir klagen
  2. (Evangelist, Jesus): Da Jesus diese Rede vollendet
        hatte
  3. Choral: Herzliebster Jesu, was hast du verbrochen
  4. (Evangelist): Da versammelten sich die Hohenpriester
  5. (Chori): Ja nicht auf das Fest
  6. (Evangelista): Da nun Jesus war zu Bethanien
  7. (Chorus): Wozu dienet dieser Unrat
  8. (Evangelista, Jesus): Da das Jesus merkte
  9. (Evangelist, Jesus, Judas):  Und er kam und fand sie
       aber schlafend
  10. Aria: So ist mein Jesus nun gefangen
  11. (Chori): Sind Blitze, sind Donner in Wolken verschwunden
```

L'attribut `format` indique au processeur le type de numérotation souhaité. La valeur est une chaîne de caractères qui est composée de caractères alphanumériques et de caractères de ponctuation. Les caractères alphanumériques spécifient le format à appliquer :

- `1` : utilise des nombres simples : 1, 2, 3, … ;
- `a` : utilise l'alphabet en minuscule : a, b, c, … ;
- `A` : utilise l'alphabet en majuscule : A, B ,C, … ;
- `i` : utilise une numérotation romaine en minuscule : i, ii, ii, iv, … ;
- `I` : utilise une numérotation romaine en majuscule : I, II, III, IV, …

Dans le programme *Number1.xsl*, nous avons utilisé la fonction *position*() pour calculer la position du nœud à numéroter. L'instruction `xsl:number` nous a uniquement servi à formater le résultat et on aurait pu obtenir le même résultat en utilisant la fonction *format-number*(). Un inconvénient de cette solution est que le compteur calculé dépend du contexte. Voici une version modifiée de la règle de transformation pour un disque :

Exemple .21 *Number1aRule.xsl* : *Sélection des séquences contenant le mot « Jesu »*

```
  <xsl:template match="DISQUE">
    <xsl:value-of select="concat(OEUVRE, $sautLigne, AUTEUR, $sautLigne)"/>
    <xsl:for-each select="CD/PLAGE/S[contains(.,'Jesu')]">
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="concat(., $sautLigne)"/>
    </xsl:for-each>
  </xsl:template>
```

Cette règle sélectionne uniquement les séquences contenant le mot « Jesu » :

Exemple .22 *Number1a.txt* : *Résultat obtenu en appliquant la règle précédente*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  1. (Evangelist, Jesus): Da Jesus diese Rede vollendet
        hatte
  2. Choral: Herzliebster Jesu, was hast du verbrochen
  3. (Evangelista): Da nun Jesus war zu Bethanien
  4. (Evangelista, Jesus): Da das Jesus merkte
  5. (Evangelist, Jesus, Judas):  Und er kam und fand sie
       aber schlafend
  6. Aria: So ist mein Jesus nun gefangen
```

Les lignes sont numérotées de 1 à 6, ce qui correspond à la position de chaque séquence dans le contexte de l'expression XPath de la boucle `xsl:for-each`. Pour obtenir le « bon » numéro pour chaque séquence il ne faut pas utiliser la fonction *position*() mais compter les prédécesseurs (axe `preceding`) de type `S` et ajouter 1 à la valeur obtenue :

```
<xsl :number value="count(preceding : :S)+1" format="1. "/>
```

On obtient alors le résultat suivant :

Exemple .23 *Number1c.txt* : *Résultat obtenu en comptant les prédécesseurs*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  2. (Evangelist, Jesus): Da Jesus diese Rede vollendet
        hatte
  3. Choral: Herzliebster Jesu, was hast du verbrochen
  6. (Evangelista): Da nun Jesus war zu Bethanien
  8. (Evangelista, Jesus): Da das Jesus merkte
  9. (Evangelist, Jesus, Judas):  Und er kam und fand sie
       aber schlafend
  10. Aria: So ist mein Jesus nun gefangen
```

Une autre possibilité est d'utiliser l'attribut `count` à la place de l'attribut `value` dans l'instruction `xsl:number` :

```
<xsl :number count="S" level="any" format="1. "/>
```

L'attribut `count` indique qu'on veut compter les nœuds de type `S` qui se trouvent avant le *nœud cible* (la plupart du temps le nœud cible correspond au nœud contexte, mais ce n'est pas toujours le cas comme nous allons le voir plus loin). Il existe deux possibilités pour choisir les nœuds qui se trouvent avant :

1. On veut compter les nœuds précédents qui ont le même père. Ceci correspond à l'utilisation de l'axe `preceding-sibling` et se traduit par la valeur `single` pour l'attribut `level`.
2. On veut compter les nœuds précédents *à n'importe quel niveau* ce qui est le cas dans notre exemple. Ceci se traduit par l'utilisation de l'axe `preceding` ou, d'une manière équivalente, par la valeur `any` pour l'attribut `level`.

Remarque : Comme pour les axes `preceding` et `preceding-sibling`, les nœuds de type **Attr** ou `Namespace` sont ignorés pendant la numérotation.

La valeur `single` pour l'attribut `level` permet d'obtenir une numérotation non plus absolue, mais relative à d'autres éléments. Dans le document *Number1d.txt* chaque séquence est numérotée par sa position dans la plage.

Exemple .24 *Number1d.txt* : *Séquences numérotées à l'intérieur de chaque plage*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  a. (Evangelist, Jesus): Da Jesus diese Rede vollendet
        hatte
```

```
  a. Choral: Herzliebster Jesu, was hast du verbrochen
  c. (Evangelista): Da nun Jesus war zu Bethanien
  e. (Evangelista, Jesus): Da das Jesus merkte
  a. (Evangelist, Jesus, Judas):  Und er kam und fand sie
        aber schlafend
  a. Aria: So ist mein Jesus nun gefangen
```

On obtient ce résultat avec l'instruction suivante :
`<xsl :number level="single" count="S" format="a. "/>`

L'attribut `from` permet de choisir une borne à partir laquelle on veut compter. Dans le document *Number1e.txt* ci-dessous chaque séquence obtient un numéro qui correspond à sa position dans le CD :

Exemple .25 *Number1e.txt* : *Séquences numérotées à l'intérieur de chaque CD*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  II. (Evangelist, Jesus): Da Jesus diese Rede vollendet
         hatte
  III. Choral: Herzliebster Jesu, was hast du verbrochen
  VI. (Evangelista): Da nun Jesus war zu Bethanien
  VIII. (Evangelista, Jesus): Da das Jesus merkte
  I. (Evangelist, Jesus, Judas):  Und er kam und fand sie
        aber schlafend
  II. Aria: So ist mein Jesus nun gefangen
```

Pour obtenir ce résultat on compte le nombre de prédécesseurs (à n'importe quel niveau) jusqu'au *premier* prédécesseur de type `CD` avec l'instruction suivante : `<xsl :number level="any" count="S" from="CD" format="I. "/>`

Tous les attributs dans l'instruction `xsl:number` sont optionnels. Les valeurs par défaut sont données dans le tableau 3.

Attribut	**Valeur par défaut**
`level`	`"single"`
`count`	un *pattern* qui sélectionne les nœuds du même type et, si disponible, du même nom que le nœud contexte
`from`	un *pattern* qui retourne faux pour tous les nœuds (par exemple `*[false()]`)
`format`	`"1"`

TAB. 3 – Valeurs par défaut pour l'instruction `xsl:number`

Regardons maintenant le cas où on ne compte pas les éléments du même type que le nœud contexte. Voici une règle qui utilise deux compteurs :

Exemple .26 *Number1fRegle.xsl* : *Règle avec deux compteurs*

```
<xsl:template match="DISQUE">
  <xsl:value-of
      select="concat(OEUVRE, $sautLigne, AUTEUR, $sautLigne)"/>
  <xsl:for-each select="CD/PLAGE/S[contains(.,'Jesu')]">
      <xsl:number level="single" count="CD" format="A-"/>
      <xsl:number level="any" count="S" from="CD" format="1. "/>
      <xsl:value-of select="concat(., $sautLigne)"/>
  </xsl:for-each>
</xsl:template>
```

Le fichier *Number1f.txt* ci-dessous montre le résultat de la règle précédente. Le premier compteur compte tous les prédécesseurs de type `CD` du nœud contexte. Si on regarde de plus près, on se rend compte que l'attribut `level` est égal à `single`, ce qui veut dire qu'on compte seulement les nœuds `CD` qui ont le même père que le nœud cible. Dans ce cas, si on prenait le nœud contexte comme nœud cible, cette instruction retournerait zéro (les éléments de type `S` n'ont pas de frère de type `CD`). En effet, avant de commencer à compter, l'instruction `xsl:number` cherche d'abord *le premier ancêtre* qui satisfait le *pattern* de l'attribut `count` et choisit ce nœud comme nœud cible. Dans notre exemple, c'est le premier ancêtre de type `CD` qui satisfait cette condition et sera choisi comme nœud cible pour le comptage.

Exemple .27 *Number1f.txt* : *Résultat en appliquant la règle précédente avec deux compteurs*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  A-2. (Evangelist, Jesus): Da Jesus diese Rede vollendet
         hatte
  A-3. Choral: Herzliebster Jesu, was hast du verbrochen
  A-6. (Evangelista): Da nun Jesus war zu Bethanien
  A-8. (Evangelista, Jesus): Da das Jesus merkte
  B-1. (Evangelist, Jesus, Judas):  Und er kam und fand sie
        aber schlafend
  B-2. Aria: So ist mein Jesus nun gefangen
```

Numérotation hiérarchique

La règle précédente *Number1fRegle.xsl* utilise deux compteurs indépendants pour compter d'abord les disques et ensuite les séquences sur chaque disque. Le résultat est un compteur hiérarchique qui associe à chaque séquence un numéro de la forme `X-n` indiquant qu'il s'agit de la `n`-ième séquence du `X`-ième disque.

L'instruction `xsl:number` permet de générer ce type de compteur avec une seule instruction en utilisant l'attribut `format` pour indiquer la numérotation d'une hiérarchie. Voici un exemple établissant une numérotation sur la hiérarchie `CD/PLAGE/S`.

Exemple .28 *Number2.xsl* : *Numérotation hiérarchique*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"
         encoding="ISO-8859-1"/>

  <xsl:variable name="sautLigne"><xsl:text>
  </xsl:text></xsl:variable>
  <xsl:variable name="espace">
    <xsl:text> </xsl:text>
  </xsl:variable>

  <xsl:template match="DISQUE">
    <xsl:value-of
        select="concat(OEUVRE, $sautLigne, AUTEUR, $sautLigne)"/>

    <xsl:for-each select="//S">
     <xsl:number count="CD|PLAGE|S" level="multiple" format="A.1.a "/>

     <xsl:value-of select="concat(., $sautLigne)"/>
    </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

Et voici le résultat obtenu.

Exemple .29 *Number2.txt* : *Résultat de la numérotation hiérarchique*

```
Die Matthäus-Passion
  Johann-Sebastian Bach
  A.1.a Chorus: Kommt, ihr Töchter, helft mir klagen
  A.2.a (Evangelist, Jesus): Da Jesus diese Rede vollendet
         hatte
  A.3.a Choral: Herzliebster Jesu, was hast du verbrochen
  A.4.a (Evangelist): Da versammelten sich die Hohenpriester
  A.4.b (Chori): Ja nicht auf das Fest
  A.4.c (Evangelista): Da nun Jesus war zu Bethanien
  A.4.d (Chorus): Wozu dienet dieser Unrat
  A.4.e (Evangelista, Jesus): Da das Jesus merkte
  B.1.a (Evangelist, Jesus, Judas):  Und er kam und fand sie
        aber schlafend
  B.2.a Aria: So ist mein Jesus nun gefangen
  B.2.b (Chori): Sind Blitze, sind Donner in Wolken verschwunden
```

Pour comprendre comment les compteurs hiérarchiques sont générés il faut introduire l'axe `ancestor`. Au lieu de compter les prédécesseurs du nœud cible, on crée d'abord une liste de tous les ancêtres du nœud contexte qui satisfont le *pattern* de l'attribut `count`. Dans notre cas cette liste contient pour chaque séquence un élément de type `S`, un élément de type `CD` et un élément de type `PLAGE`. Cette liste est ordonnée par rapport au niveau de l'ancêtre dans l'arbre (l'élément de type `CD` est avant l'élément de type `PLAGE` qui est à son tour avant l'élément de type `S`). Comme pour les compteurs simples, la création de cette liste s'arrête quand on tombe sur un ancêtre qui satisfait le *pattern* de l'attribut `from`.

La longueur de la liste d'ancêtres obtenue par cette première étape correspond au nombre de compteurs simples du compteur hiérarchique : chaque nœud de la liste est choisi comme nœud cible pour un compteur simple qui compte le nombre de *frères précédents* qui satisfont à leur tour le pattern spécifié par l'attribut `pattern`.

Les compteurs obtenus sont formatés par rapport à la valeur de l'attribut `format`. Si le nombre de compteurs simples est supérieur au nombre de caractères alphanumériques dans l'attribut format, les caractères sont ignorés de droite vers la gauche.

Le document *Thesaurus.xml* est un autre exemple illustrant l'utilisation d'un compteur hiérarchique.

Exemple .30 *Thesaurus.xml* : *Classification des vertébrés*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="Thesaurus.xsl" type="text/xsl"?>
<BRANCHE nom="Vertébrés">
  <CLASSE nom="Mammifères">
    <ORDRE nom="Primates">
      <GENRE nom="Singes">
        <ESPECE nom="Ouistiti"/>
        <ESPECE nom="Gorille"/>
      </GENRE>
      <GENRE nom="Homme"/>
    </ORDRE>
  </CLASSE>
  <CLASSE nom="Poissons">
    <ORDRE nom="Téléostéens">
      <GENRE nom="Perciformes">
        <ESPECE nom="Perche"/>
        <ESPECE nom="Dorade"/>
      </GENRE>
    </ORDRE>
  </CLASSE>
</BRANCHE>
```

Exemple .31 *Thesaurus.xsl* : *Compteur hiérarchique*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text" encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <xsl:text>Les vertébrés:&#xA;</xsl:text>
    <xsl:apply-templates select="BRANCHE/*"/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:number count="BRANCHE//*" level="multiple" format="- A.a.1 "/>
    <xsl:value-of select="concat(@nom,'&#xA;')"/>
    <xsl:apply-templates select="*"/>
  </xsl:template>
</xsl:stylesheet>
```

Le compteur hiérarchique du programme *Thesaurus.xsl* compte tous les descendants de l'élément racine. Le premier niveau du compteur correspond aux classes dans la branche des vertébrés (éléments CLASSE) et utilise l'alphabet en majuscule pour la numérotation. Les différents ordres (niveau 2) sont numérotés par l'alphabet en minuscule. Tous les autres niveaux sont numérotés en utilisant le système décimal.

Exemple .32 *Thesaurus.txt* : *Résultat de la transformation du document Thesaurus.xml*

```
Les vertébrés:
- A Mammifères
- A.a Primates
- A.a.1 Singes
- A.a.1.1 Ouistiti
- A.a.1.2 Gorille
- A.a.2 Homme
- B Poissons
- B.a Téléostéens
- B.a.1 Perciformes
- B.a.1.1 Perche
- B.a.1.2 Dorade
```

Les autres attributs

Voici, pour conclure, les autres attributs de `xsl:number` :

1. `lang` : le code de la langue utilisée pour déterminer l'alphabet ;
2. `letter-value` : peut valoir `alphabetic` (le défaut) ou `traditional` pour choisir un type de numérotation dans des langues qui permettent plusieurs séquences possibles avec le même alphabet de compteurs (par exemple l'Hébreux) ;
3. `grouping-separator` : les chiffres de grande taille peuvent être séparés en groupes pour plus de lisibilité (par exemple 1000000 s'écrit plus lisiblement 1 000 000) : l'attribut indique le séparateur des groupes (par exemple l'espace, ou la virgule) ;
4. `grouping-size` : même préoccupation que précédemment, mais cet attribut indique la taille de chaque groupe (par exemple 3).

xsl:otherwise

Syntaxe

```
<xsl :otherwise>
        corps de règle
</xsl :otherwise>
```

Description

Cet élément est subordonné à l'élément `xsl:choose` : voir page **8**.

xsl:output

Syntaxe

```
<xsl :output
    method="xml" | "html" | "text" | QName
    version=NMToken
    encoding=CData
    omit-xml-declaration=("yes" | "no")
    standalone=("yes" | "no")
    doctype-public=CData
    doctype-system=URI
    cdata-section-elements=QNames
    indent=("yes" | "no")
    media-type=CData />
```

Description

Cet élément indique au processeur le type de document à produire. En fonction de ce type, des mises en forme spéciales peuvent être appliquées. Les trois types de document préconisés par la norme sont `html`, `xml` et `text`, la valeur par défaut étant `xml`. De plus d'autres types de document spécifiques à un processeur peuvent être pris en compte, mais le programme ne sera alors pas portable.

Les attributs `version` et `encoding` correspondent à la version XML (à l'heure actuelle la 1.0) ou HTML (à l'heure actuelle la 4.0), et à l'encodage du document résultat.

L'attribut `omit-xml-declaration` (avec pour valeur par défaut `no`) peut être utilisé pour supprimer la déclaration XML dans le résultat (sa valeur doit alors être `yes`). L'attribut `standalone` est spécifique à un document résultat de type XML : il indique si le document fait référence ou non à des entités externes.

Les attributs `doctype-public` et `doctype-system` peuvent être utilisés pour générer un document XML avec une DTD. Ces attributs permettent d'inclure une déclaration de DTD dans le prologue du document résultat. Si, par exemple, l'élément `xsl:output` est le suivant :

```
<xsl :output method='html'
doctype-public='-//W3C//DTD HTML 4.0//EN'/>
```

on obtiendra la déclaration de type dans le document :

```
<!DOCTYPE html PUBLIC '-//W3C//DTD HTML 4.0//EN'/>
```

Si on utilise l'attribut `doctype-system`, avec comme valeur l'URL du fichier contenant la DTD, la déclaration d'une DTD XML sera incluse dans le document avec `SYSTEM` suivi de l'URL à la place de `PUBLIC`.

On peut demander à ce que les fils de type **Text** de certains éléments dans le document XML résultat fassent partie d'une section littérale (`CDATA`). Dans ce cas les types de ces éléments doivent être donnés, séparés par des espaces, dans la valeur de l'attribut `cdata-section-elements`.

Le document résultat peut être indenté pour augmenter la lisibilité du résultat si l'attribut `indent` est à `yes` (ce qui n'est pas le cas par défaut). Enfin l'attribut `media-type` donne le type MIME du document produit. Ce type peut être utile quand le document est transmis par le protocole HTTP. Pour HTML il doit valoir `text/html`, pour WML `text/xml`, etc. Sa valeur par défaut est `text/xml`.

Exemples

Voir page **??** la description de cet élément et son application à une transformation XHTML → HTML.

xsl:param

Syntaxe

```
<xsl :param
    name=QName
    select=Expression>
        corps de règle
</xsl :param>
```

Description

Cet élément peut être utilisé de deux manières :

1. dans une règle, pour indiquer un paramètre local reconnu et pris en compte par la règle ;
2. comme élément de premier niveau, pour désigner des paramètres extérieurs passés au programme.

L'attribut `select` est optionnel. S'il est présent, il donne la valeur par défaut du paramètre. Si `select` est omis, le corps de l'élément (également optionnel) `xsl:param` définit la valeur par défaut du paramètre.

Exemples

Voir la section consacrée au passage de paramètres, page **??**.

xsl:preserve-space

Syntaxe

```
<xsl :preserve-space
    elements=QNames/>
```

Description

Cet élément, ainsi que son complément `xsl:strip-space`, permet de contrôler la prise en compte des nœuds de type **Text** constitués uniquement de blancs (« nœuds blancs »). Il est associé à une liste de noms d'éléments donnée dans l'attribut `elements` (le caractère `*` indiquant tous les éléments). Pour tous les élément indiqués, le nœuds de type **Text** doivent être préservés même s'ils ne contiennent que des espaces.

Par défaut le processeur XSLT doit conserver les nœuds blancs, ce qui peut introduire des effets désagréables, notamment quand on utilise la fonction *position*(). Tout se passe donc comme si l'option suivante était prise par défaut :

```
<xsl :preserve-space elements="*"/>
```

Exemples

L'exemple suivant est un document avec l'indentation habituelle pour clarifier la hiérarchie des éléments. <A> et <B> ont tous deux deux fils de type **Text** vides, encadrant un élément.

Exemple .33 *Space.xml* : *Un document XML avec indentation*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<A>
   <B>
      <C>Contenu</C>
   </B>
</A>
```

Le programme suivant commence par définir une règle inverse à l'option par défaut : tous les nœuds de type **Text** constitués uniquement d'espaces doivent être supprimés. On procède ensuite par exception en indiquant que ce type de nœud doit être préservé uniquement pour les éléments de type B.

Exemple .34 *PreserveSpace.xsl* : *Exemple de* `xsl:preserve-space` *et* `xsl:strip-space`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:strip-space elements="*"/>
  <xsl:preserve-space elements="B"/>

  <xsl:template match="/A">
      Fils de A : <xsl:value-of select="count(node())"/>
      <xsl:apply-templates select="B"/>
  </xsl:template>

  <xsl:template match="B">
      Fils de B : <xsl:value-of select="count(node())"/>
  </xsl:template>

</xsl:stylesheet>
```

On appelle ensuite deux règles, l'une pour les éléments de type A, l'autre pour les éléments de type B, et dans chaque règle on affiche le nombre de fils du nœud courant. On obtient 1 pour <A> et 3 pour <B>.

xsl:processing-instruction

Syntaxe

```
<xsl :processing-intruction
    name=QName>
        corps de règle
</xsl :processing-instruction>
```

Description

Cette instruction permet d'insérer un nœud de type **ProcessingInstruction** dans le document résultat. L'attribut `name` est obligatoire et donne le nom ou « cible » de l'instruction de traitement. Le contenu de l'élément `xsl:processing-instruction` doit produire du texte formant le contenu de l'instruction de traitement dans le document résultat.

Syntaxe

```
<xsl :result-document
    href=URI
    format= ("xml" | "html" | "xhtml" | "text" | autre)>
    corps de règle </xsl :result-document>
```

Description

La version 1.0 de XSLT ne permet de produire qu'un seul document par transformation. Cette instruction est introduite dans la version 2.0 afin de pouvoir engendrer plusieurs documents à partir d'une seule transformation. Elle est également proposée comme extension par la plupart des processeurs XSLT actuels, qui sont conformes à la version 1.0. Ces extensions s'appellent `saxon :output` dans le processeur Saxon et `xalan :write` dans le processeur Xalan.

L'attribut obligatoire `href` permet de spécifier l'URL du document à générer. Voir instruction `xsl:output` pour la signification des autres attributs qui permettent de contrôler la sérialisation du résultat.

Le résultat du corps de règle est écrit dans le fichier désigné par l'attribut `href` et le processeur XSLT doit évidemment avoir les droits d'écriture nécessaires. Pour cette raison l'URL prend généralement la forme d'un chemin sur le disque local en utilisant le protocole `file :`.

Les chemins relatifs prennent comme point de départ le répertoire du document de sortie au moment où le processeur à rencontré l'instruction `xsl:document`. Ceci a plusieurs conséquences :

- Dans un environnement comme Cocoon, qui transforme les documents XML dynamiquement, l'emplacement peut être un répertoire temporaire extérieur à l' espace visible par le serveur web. Dans ce cas il est conseillé d'utiliser des chemins absolus.
- Les appels d'instruction `xsl:document` peuvent être imbriqués et la même instruction peut engendrer des documents à différents endroits (voir exemple).

L'attribut `format` permet d'indiquer le format de sérialisation du fragment généré (`xml`, `html`, `xhtml` ou `text`).

Exemples

Le document *Equipes.xml* contient deux équipes avec leurs membres.

Exemple .35 *Equipes.xml* : *Équipes avec membres*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<EQUIPES>
  <EQUIPE NUMERO="1">
    <MEMBRE NOM="Scholl"/>
    <MEMBRE NOM="Vodislav"/>
    <MEMBRE NOM="Gross"/>
    <MEMBRE NOM="Amann"/>
  </EQUIPE>
  <EQUIPE NUMERO="2">
    <MEMBRE NOM="Spyratos"/>
    <MEMBRE NOM="Rigaux"/>
  </EQUIPE>
</EQUIPES>
```

Le programme *SiteEquipes.xsl* crée pour chaque équipe une page *index.html* dans le sous-répertoire `EQUIPE`*Num* où le paramètre *Num* correspond au numéro de l'équipe. La règle de transformation des membres est appelée avant la fermeture de l'instruction `xsl:document` et crée une page pour chaque membre dans le même répertoire. Par une seule transformation du fichier *Equipes.xml* on obtient ainsi huit fichiers dans deux répertoires différents :

- répertoire *EQUIPE1* : *index.html*, *Scholl.html*, *Vodislav.html*, *Gross.html*, *Amann.html* ;

– répertoire *EQUIPE2* : *index.html*, *Spyratos.html*, *Rigaux.html*.

Exemple .36 *SiteEquipe.xsl* : *Génération de pages HTML*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="EQUIPE">

  <!-- génération de la page de l'équipe -->
  <xsl:result-document
    href="EQUIPE{@NUMERO}/index.html"
    format="html">
    <html>
      <!-- page de l'equipe -->
    </html>

    <!-- génération d'une page pour chaque membre -->
    <xsl:apply-templates select="MEMBRE"/>
  </xsl:result-document>

</xsl:template>

<xsl:template match="MEMBRE">
  <xsl:result-document
    href="{@NOM}.html"
    format="html">
    <html>
      <!-- page de membre -->
    </html>
  </xsl:result-document>
</xsl:template>

</xsl:stylesheet>
```

Voir également page **??** et page **??** d'autres exemples avec Xalan et Saxon.

xsl:sort

Syntaxe

```
<xsl :sort
    select=expression
    lang=CodeLangue
    order=("ascending" | "descending")
    case=("lower-first" | "upper-first")
    data-type= ("text" | "number" ) />
```

Description

Cet élément est utilisé pour trier les nœuds sélectionnés soit par un appel de règles `xsl:apply-templates`, soit par une boucle `xsl:for-each`. L'élément `xsl:sort` doit toujours apparaître immédiatement après la balise ouvrante de `xsl:for-each` ou `xsl:apply-templates`, le mélange avec des éléments `xsl:with-param` étant possible dans le second cas.

La clé de tri est indiquée par l'expression de l'attribut `select` (tous les attributs sont optionnels). Cette expression est évaluée en prenant pour nœud courant, tour à tour, chaque nœud de l'ensemble sélectionné

par `xsl:for-each` ou `xsl:apply-templates`. Le tri de l'ensemble s'effectue alors d'après les valeurs obtenues.

Par défaut le tri se fait d'après l'ordre lexicographique sur les chaînes de caractères, ce qui signifie que « 4 » est plus grand que « 30 » (les caractères sont comparés de gauche à droite). En indiquant la valeur `number` pour l'attribut `data-type`, on obtient un classement numérique.

L'ordre de tri est croissant par défaut et peut être modifié par l'attribut `order`. L'attribut `lang` est supposé indiquer l'ordre des caractères spéciaux (est-ce que « é » est inférieur ou supérieur à « è » ?) qui dépend de la langue. Enfin `case` indique le classement des mots en majuscule par rapport aux mêmes mots en minuscules.

On peut indiquer plusieurs `xsl:sort` successifs. Le processeur effectue un premier tri avec la première clé, puis un second tri sur les groupes qui n'ont pas été départagés par le premier tri, et ainsi de suite.

Exemples

Le tri est présenté page **??**. Voici un exemple complémentaire montrant comment trier les chapitres du document *XBook.xml* (page 8) d'après leur nombre de pages.

Exemple .37 *RefSort.xsl* : *Exemple de* `xsl:sort`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" encoding="ISO-8859-1"/>

  <xsl:template match="/">
     <CHAPITRES-TRIES>
      <xsl:apply-templates select="//CHAPITRE">
        <xsl:sort select="@LONGUEUR" data-type="number"/>
      </xsl:apply-templates>
     </CHAPITRES-TRIES>
  </xsl:template>

  <xsl:template match="CHAPITRE">
     <CHAPITRE>
       <TITRE><xsl:value-of select="TITRE"/></TITRE>
       <LONGUEUR><xsl:value-of select="@LONGUEUR"/></LONGUEUR>
     </CHAPITRE>
  </xsl:template>
</xsl:stylesheet>
```

Remarquez l'attribut `data-type` avec la valeur `number`. Par défaut, le classement lexicographique dirait que l'avant-propos (8 pages) est plus long que tous les autres chapitres.

Toute valeur associée à un nœud par une expression XPath peut être utilisée comme critère de tri, y compris celles basées sur la fonction *document*(). Reprenons le document *Films.xml*, page 16, et supposons que le fichier *Metteurs.xml* contienne les informations sur les metteurs en scènes.

Exemple .38 *Metteurs.xml* : *Les metteurs en scène*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<ARTISTES>
  <ARTISTE><NOM>Scott</NOM>
          <PRENOM>Ridley</PRENOM>
          <ANNEE>1943</ANNEE>
  </ARTISTE>
  <ARTISTE><NOM>Hitchcock</NOM>
```

```
          <PRENOM>Alfred</PRENOM>
          <ANNEE>1899</ANNEE>
  </ARTISTE>
  <ARTISTE><NOM>Kurosawa</NOM>
          <PRENOM>Akira</PRENOM>
          <ANNEE>1910</ANNEE>
  </ARTISTE>
  <ARTISTE><NOM>Woo</NOM>
          <PRENOM>John</PRENOM>
          <ANNEE>1946</ANNEE>
  </ARTISTE>
  <ARTISTE><NOM>Cameron</NOM>
          <PRENOM>James</PRENOM>
          <ANNEE>1954</ANNEE>
  </ARTISTE>
  <ARTISTE><NOM>Tarkovski</NOM>
          <PRENOM>Andrei</PRENOM>
          <ANNEE>1932</ANNEE>
  </ARTISTE>
</ARTISTES>
```

Voici un programme qui trie les films en fonction de l'année de naissance de leur metteur en scène.

Exemple .39 *RefSort2.xsl* : *Films triés sur l'année de naissance du réalisateur*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" encoding="ISO-8859-1"/>

  <xsl:variable name="artistes"
             select="document('Artistes.xml')/ARTISTES"/>

  <xsl:template match="FILMS">
     <FILMS-TRIES>
      <xsl:apply-templates select="FILM">
        <xsl:sort
           select="$artistes/ARTISTE[NOM = current()/AUTEUR]/ANNEE"
           data-type="number"/>
      </xsl:apply-templates>
     </FILMS-TRIES>
  </xsl:template>

  <xsl:template match="FILM">
    <FILM>
     <TITRE><xsl:value-of select="TITRE"/></TITRE>
     <AUTEUR><xsl:value-of select="AUTEUR"/></AUTEUR>
     <NE-EN>
        <xsl:value-of
           select="$artistes/ARTISTE[NOM = current()/AUTEUR]/ANNEE"/>
      </NE-EN>
    </FILM>
  </xsl:template>
</xsl:stylesheet>
```

La variable `$artistes` est initialisée avec le contenu du document *Metteurs.xml* (voir la description de la fonction *document*() page **39**). À partir de là on peut l'utiliser comme s'il s'agissait de l'élément racine de ce document. L'expression :

```
$artistes/ARTISTE[NOM = current()/AUTEUR]/ANNEE
```

sélectionne l'élément <ARTISTE> dont l'élément fils <NOM> a un contenu identique à celui du nœud courant. **Attention** : il s'agit de l'un des rares cas où le nœud contexte de XPath est différent du nœud courant de XSLT.

1. le nœud courant est un des éléments de type FILM du document *Films.xml* ;
2. le nœud contexte dépend des étapes de l'expression XPath : dans l'expression ci-dessus, si on remplaçait *current*() par « . », on désignerait un élément <ARTISTE> du document *Metteurs.xml*, et pas un film.

La fonction *current*() désigne toujours l'élément courant (voir page **38**).

xsl:strip-space

Syntaxe

```
<xsl :strip-space
    elements=QNames/>
```

Description

Cet élément donne la liste des éléments du documents source dont les fils de type **Text** constitués uniquement d'espaces doivent être supprimés. L'attribut `elements` est obligatoire : il indique, séparés par des blancs, la liste des éléments concernés par `xsl:strip-space`. On peut utiliser le caractère « * » pour désigner *tous* les éléments.

Exemples

Voir page **??**, ainsi que la description de `xsl:preserve-space`, page **27**.

xsl:stylesheet

Syntaxe

```
<xsl :stylesheet
    id=idElément
    extension-elements-prefixes=tokens
    exclude-result-prefixes=tokens
    version=number>
        éléments de premier niveau
</xsl :stylesheet>
```

Description

Cet élément est toujours l'élément racine d'un programme XSLT (`xsl:transform` est un synonyme). Il doit toujours être accompagné d'un numéro de version (actuellement seule la 1.0 est une recommandation du W3C) afin de permettre au processeur de s'adapter aux éventuelles différences entre les versions successives de XSLT. Tous les autres attributs sont optionnels.

L'attribut `id` donne un identifiant au programme. Cet attribut n'est utile que quand le programme est inclus directement dans un document XML, et permet alors d'identifier le nœud racine des instructions XSLT.

L'attribut `extension-elements-prefixes` donne la liste des préfixes des éléments qui sont des extensions de XSLT. On peut exclure les éléments de certains espaces de noms en les indiquant dans l'attribut `exclude-result-prefixes`.

Les fils de `xsl:stylesheet` sont tous les éléments de premier niveau de XSLT (voir tableau **??**, page **??**), les éléments de type `xsl:import` devant apparaître en premier.

Exemples

Voir notamment la section consacrée à la structure d'un document XSLT, page **??**, et aux espaces de noms, page **??**.

xsl:template

Syntaxe

```
<xsl :template
    name=QName
    match=Pattern
    mode=QName
    priority=number>
        (<xsl :param>*, corps de règle)
</xsl :template>
```

Description

Cet élément définit une *règle* XSLT. Une règle peut être appelée par son nom (attribut `name`) avec une instruction `xsl:call-template`. On peut aussi l'associer à un *pattern* avec l'attribut `match`, et la règle est déclenchée pour tous les nœuds qui satisfont le *pattern*. L'un des deux attributs doit être défini, mais pas les deux à la fois.

Deux attributs complémentaires conditionnent le choix d'une règle. L'attribut `priority` détermine le degré de priorité : il est pris en compte quand plusieurs règles sont possibles pour un `xsl:apply-templates`. L'attribut `mode` permet de définir plusieurs catégories de règles à appliquer dans des contextes différents.

Exemple

Voir la section du chapitre **??** consacrée aux règles, page **??**.

xsl:text

Syntaxe

```
<xsl :text
    disable-output-escaping=("yes" | "no")>
        CData
</xsl :text>
```

Description

Cette instruction permet d'insérer un nœud de type **Text** dans le document résultat. Le contenu de l'élément ne doit pas contenir de balisage : les caractères spéciaux doivent donc être référencés *via* une entité prédéfinie comme `<` pour '<' ou `&` pour '&'. Par défaut tous ces caractères spéciaux seront à leur tour produits sous forme de référence à une entité dans le document résultat, mécanisme désigné par le terme *escaping* en anglais.

Pour insérer directement le caractère, et pas la référence à l'entité, on peut mettre à `yes` l'attribut `disable-output-escaping`. Attention : cela affecte le document résultat, et pas le programme XSLT qui doit, lui, toujours utiliser les références aux entités.

Les blancs introduits dans le contenu d'un élément `xsl:text` sont toujours préservés dans le document résultat. Voir également page **??** pour la gestion des nœuds de texte dans le document résultat.

Exemples

Le programme XSLT ci-dessous produit une simple phrase contenant quelques caractères réservés. L'utilisation des entités est requise.

Exemple .40 *Text.xsl* : *Exemple de* `xsl:text`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">

    <xsl:text disable-output-escaping="yes">
      Si  0 &lt; 1, alors 1 &gt; 0,
         &amp; vice et versa
    </xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Le résultat de ce programme est le suivant. Il contient des caractères réservés qui le rendent impropre à une inclusion dans un document XML bien formé.

```
    Si 0 < 1, alors 1 > 0,
& vice et versa
```

Si `disable-output-escaping` n'était pas défini (ou défini à `no`, ce qui revient au même), on obtiendrait dans le résultat une version avec entités identique à celle du programme XSLT.

xsl:transform

Syntaxe

```
<xsl :transform
    id=idElément
    extension-elements-prefixes=tokens
    exclude-result-prefixes=tokens
    version=number>
        éléments de premier niveau
</xsl :transform>
```

Description

Cet élément est synonyme de `xsl:stylesheet` : voir page **33**.

xsl:value-of

Syntaxe

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

Description

Cet élément évalue une expression sur le document source (ou sur un document référencé par la fonction *document*()), puis convertit le résultat en chaîne de caractères selon les règles de conversion présentées dans le chapitre **??**, et enfin insère la chaîne dans le document résultat.

L'attribut `disable-output-escaping` a la même signification que dans le cas de l'élément `xsl:text` décrit précédemment.

Exemples

Voir page **??**.

xsl:variable

Syntaxe

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

Description

Une *variable* dans XSLT est un nom référençant une information qui peut être de nature très variée : un nœud, un ensemble de nœuds, une valeur numérique ou alphanumérique, ou un *result tree fragment*, type spécifique à XSLT 1.0 qui va probablement disparaître avec XSLT 2.0.

La *valeur* d'une variable est définie soit par l'évaluation de l'expression associée à l'attribut `select` (auquel cas le contenu doit être vide), soit par le contenu de l'élément `xsl:variable` s'il n'y a pas de `select`.

La *portée* d'une variable dépend de la position de l'élément dans le programme. Si c'est un élément de premier niveau (fils de `xsl:stylesheet`) elle définit une *variable globale* qui est visible dans tout le programme XSLT. Si, en revanche, l'instruction apparaît dans un élément `xsl:template`, elle définit une *variable locale* qui est visible dans tous les frères droits de `xsl:variable` et leurs descendants (une variable locale est seulement visible à l'intérieur d'une règle).

On ne peut pas définir une variable locale avec le même nom qu'une autre variable locale visible (autrement dit on ne peut pas trouver deux `xsl:variable` avec la même valeur pour l'attribut `name` si les éléments ont, au moins en partie, le même espace de visibilité). En revanche une variable locale peut redéfinir une variable globale.

Dans le cas où une variable globale porte le même nom qu'une variable importée, l'ordre de préséance joue et la définition de variable globale présente dans le document principal ou dans le document importé ayant la plus grande préséance est prise en compte.

Exemples

Voir la section consacrée aux variables, page **??**.

xsl:when

Syntaxe

```
<xsl :when
    test=Expression />
        corps de règle
</xsl :when>
```

Description

Cet élément est subordonné à l'élément `xsl:choose` : voir page **8**.

xsl:with-param

Syntaxe

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

Description

Cet élément est utilisé pour passer des paramètres à une règle. Il suit un `xsl:call-template`, un `xsl:apply-templates` ou un `xsl:apply-imports`.

La valeur du paramètre est déterminée, comme pour les variables, en évaluant l'attribut `select` s'il est présent, ou en prenant le contenu de l'élément sinon. Comme pour beaucoup d'autres aspects de la programmation XSLT, le passage de paramètres est très faiblement typé : on peut passer à une règle des paramètres qu'elle n'attend pas, ou au contraire ne pas lui passer de paramètres. Dans le premier cas le paramètre est tout simplement ignoré. Dans le second cas la règle prend en compte la valeur par défaut du paramètre, si cette valeur existe : voir page **34**.

Exemples

Voir la section consacrée au passage de paramètres, page **??**.

2 Fonctions XPath/XSLT

Les fonctions qui suivent sont utilisables dans toute expression XPath apparaissant dans un programme XSLT. La plupart sont des fonctions définies dans la recommandation XPath, mais certaines (*format-number*(), *document*(), *key*() par exemple) sont spécifiques à XSLT. Nous ne les distinguons pas dans ce qui suit.

Une fonction prend zéro, un ou plusieurs arguments, parfois optionnels. Nous utilisons les conventions habituelles des DTD : arg^* indique un argument qui peut être répété 0 ou plusieurs fois, $arg?$ indique un argument optionnel, etc.

En général le type de ces arguments est l'un des quatre types de base XPath (`number`, `string`, `boolean` et `node-set`), ou l'union de ces quatre types que nous dénotons `objet` comme dans la recommandation XPath.

Bien entendu un argument peut lui-même être le résultat d'une autre fonction, ou bien être une expression XPath dont l'évaluation donnera la valeur à passer à la fonction. Si le type de cette valeur ne correspond pas au type attendu, par exemple `number`, le processeur tentera d'effectuer une conversion au moment de l'appel de la fonction (voir chapitre **??**, page **??**).

boolean

`Boolean` *`boolean`* `(object` *`arg`*`)`

La fonction *boolean*() convertit son argument *arg* en booléen. Les règles de conversion ont été présentées dans le chapitre **??**, page **??**. Rappelons-les brièvement :

1. un numérique à zéro est converti à `false`, toutes les autres valeurs sont `true` ;

2. une chaîne de longueur nulle est `false`, toutes les autres sont `true` ;
3. un `node-set` vide est `false`, sinon la conversion renvoie `true`.

ceiling

`number` *`ceiling`* `(number` *`valeur`*`)`

Cette fonction renvoie le plus petit entier immédiatement supérieur à *`valeur`*. Par exemple `ceiling(1.5)` est 2.

concat

`string` *`concat`* `(string` *`chaîne1`*`, string` *`chaîne2`*`,`
`string` *`chaînes`**`)`

Cette fonction concatène les chaînes de caractères (au moins deux) qui lui sont passées en argument.

contains

`Boolean` *`contains`* `(string` *`chaîne`*`, string` *`sous-chaîne`*`)`

Cette fonction teste si *`chaîne`* contient *`sous-chaîne`*, et renvoie `true` ou `false` selon le cas.

count

`number` *`count`* `(node-set` *`nœuds`*`)`

Cette fonction compte le nombre de nœuds dans le `node-set` passé en argument.

current

`node-set` *`current`* `( )`

Cette fonction renvoie le nœud courant. Qu'est-ce que le nœud courant (qu'il faut bien distinguer du nœud contexte pour une expression XPath) ? Un programme XSLT consiste à considérer certains nœuds du document source, à rechercher quelle est la règle associée à chaque nœud et à instancier cette règle. Chaque règle s'instancie donc en association avec le nœud qui a déclenché sont instanciation : c'est le *nœud courant*.e

Au sein d'une règle on peut trouver des instructions qui vont temporairement changer le nœud courant : `xsl:apply-templates` va sélectionner, avec une expression XPath un ensemble de nœuds et les prendre chacun à son tour comme nœuds courants pour instancier un règle, et de même pour `xsl:for-each`. Ces deux éléments modifient localement le nœud courant, qui revient à sa valeur initiale quand l'évaluation de `xsl:apply-templates` ou `xsl:for-each` est terminée.

Le nœud courant peut être différent du nœud contexte, désigné dans XPath par l'expression `self :node()` ou « . ». Par exemple, dans l'expression `FILM/TITRE[.='Alien']` le nœud courant est celui à partir duquel on va chercher un fils de type `FILM`, puis un petit-fils de type `TITRE`, etc. Dans cette expression en revanche, le nœud contexte désigné par « . » est un élément de type `TITRE`. Les prédicats sont les seuls cas où le nœud courant et le nœud contexte diffèrent, et les seuls cas où l'utilisation de la fonction *current*() est justifiée. Voici par exemple un appel de règle qui sélectionne tous les nœuds frères du nœud courant qui ont le même contenu.

```
<xsl :apply-templates match="../*[.=current()]">
```

La référence de l'élément `xsl:sort` présente un exemple où l'utilisation de `current` est impérative.

document

`node-set` *`document`* `(object` *`uri1`*`, node-set uri2?)`

Cette fonction est le plus souvent utilisée en passant un seul argument, une chaîne de caractères correspondant à l'URI du document à charger. La fonction accède alors à ce document, l'analyse et renvoie le nœud racine du document.

De manière plus générale, la fonction prend comme premier argument un ensemble de nœuds (obtenu par exemple avec une expression XPath), chacun de ces nœuds étant traité comme une chaîne de caractères contenant une URI. La fonction accède alors à chacune de ces URI, et on obtient en résultat l'ensemble des racines des documents.

Quand l'URI est relative (autrement dit elle ne commence pas par quelque chose comme « / » ou « http :// »), l'URI de base est celle où se trouve le programme XSLT. En d'autres termes un appel comme `document('DOCS/doc.xml')` recherchera le fichier *doc.xml* dans le sous-répertoire *DOCS* du répertoire courant.

Cette URI de base peut être explicitement spécifiée avec le deuxième argument de *document*(). Cet argument doit être un `node-set` dont le premier est converti en chaîne et interprété comme une URI.

Enfin, il faut signaler que quand une chaîne vide est passée à la fonction, c'est le programme XSLT lui-même qui est ouvert.

L'utilisation de la fonction *document*() est décrite page **??**.

element-available

`Boolean` *`element-available`* `(string` *`nomElément`*`)`

Cette fonction teste si un élément XSLT est connu du processeur. Elle est principalement utile pour tester si certaines extensions largement répandues (comme, par exemple, `xsl:document`) sont disponibles.

false

`Boolean` *`false`* `()`

Cette fonction renvoie `false`, et peut être utilisée pour palier l'absence de constantes booléennes dans XSLT. En pratique les conversions effectuées automatiquement sont le plus souvent suffisantes, et évitent d'avoir à écrire « *`expression=false()`* » alors que « `not(`*`expression`*`)` » a une signification équivalente.

floor

`number` *`floor`* `(number` *`valeur`*`)`

Cette fonction renvoie le plus grand entier immédiatement inférieur à *`valeur`*. Par exemple `floor(1.5)` est 1. Appliquée à une valeur entière, la fonction renvoie cette valeur.

format-number

`string` *`format-number`* `(number` *`valeur`*`, string` *`format`*`, string` *`nom`*`?)`

Cette fonction permet de formater des nombres. Elle prend deux arguments : le nombre à afficher et le format. La chaîne de formatage est composée d'un préfixe optionnel, d'un *motif de formatage* et d'un suffixe optionnel. Le motif de formatage est une séquence de caractères spéciaux définis dans le tableau 4.

Le préfixe et le suffixe dans une chaîne de formatage ne doivent pas contenir des caractères spéciaux de formatage. Par exemple, la chaîne de caractères `'##,##0.00 E '` est composée du motif de formatage `##,##0.00`, suivi par le suffixe `' E '` (le préfixe est vide).

Caractère	Signification	Défaut
zero-digit	ce caractère est toujours remplacé par un chiffre	0
digit	ce caractère est remplacé par un chiffre ou effacé s'il n'y a pas de remplaçant	#
decimal-point	séparateur entre la partie entière et la fraction d'un nombre	.
grouping-separator	séparateur de groupes de chiffres	,
pattern-separator	caractère pour séparer le format des nombres positifs du format des nombres négatifs	;
minus-sign	caractère utilisé pour distinguer les nombres négatifs	-
percent-sign	caractère pourcent	%
per-mille	caractère pourmille	‰
apostrophe	symbole d'échappement pour l'utilisation des caractères spéciaux dans le résultat	'

TAB. 4 – Caractères spéciaux de formattage

Exemple .41 *Articles.xml* : *Articles avec prix en euros, remise et taux de change*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FACTURE>
  <CHANGES>
     <FF>6.55957</FF>
     <ATS>13.7603</ATS>
  </CHANGES>
  <ARTICLES>
     <ARTICLE NOM='PC Portable' PRIX='01000' />
     <ARTICLE NOM='Lecteur DVD' PRIX='767.60' />
  </ARTICLES>
  <REMISE VALEUR='0.05' />
</FACTURE>
```

Le document *Articles.xml* montre les taux de change du Franc Français (FF) et du Schilling Autrichien (ATS) en Euros. Il contient également le prix de deux articles en Euros et une remise de 5 pourcent.

Exemple .42 *Facture.xsl* : *Programme de transformation en FF et ATS*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text" encoding="ISO-8859-1"/>

  <xsl:variable name="tauxFF" select="/FACTURE/CHANGES/FF"/>
  <xsl:variable name="tauxATS" select="/FACTURE/CHANGES/ATS"/>

  <!-- insertion de blancs -->
  <xsl:template name="Blancs">
    <xsl:param name="count"/>
    <xsl:if test="$count">
      <xsl:text> </xsl:text>
      <xsl:call-template name="Blancs">
        <xsl:with-param name="count" select="$count - 1"/>
      </xsl:call-template>
    </xsl:if>
```

```
  </xsl:template>

  <xsl:template match="/">

    <!-- calcul et conversion des prix de chaque article -->
    <xsl:for-each select="/FACTURE/ARTICLES/ARTICLE">
      <xsl:value-of select="@NOM"/>
      <xsl:call-template name="Blancs">
        <xsl:with-param name="count" select="15 - string-length(@NOM)"/>
      </xsl:call-template>
      <xsl:call-template name="Conversion">
        <xsl:with-param name="prix" select="@PRIX"/>
      </xsl:call-template>
    <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>

    <!-- calcul et affichage de la remise -->
    <xsl:text>Remise </xsl:text>
    <xsl:variable name="remise" select="/FACTURE/REMISE/@VALEUR"/>
    <xsl:variable name="somme" select="sum(/FACTURE/ARTICLES/ARTICLE/@PRIX)"/>
    <xsl:value-of select="format-number($remise,' ##%     ')"/>
    <xsl:call-template name="Conversion">
      <xsl:with-param name="prix" select="-($somme * $remise)"/>
    </xsl:call-template>
    <xsl:text>&#xA;&#xA;</xsl:text>

    <!-- calcul et affichage de la somme -->
    <xsl:text>Somme          </xsl:text>
    <xsl:call-template name="Conversion">
      <xsl:with-param name="prix" select="$somme * (1-$remise)"/>
    </xsl:call-template>
  </xsl:template>

  <!-- règle de conversion -->
  <xsl:template name="Conversion">
    <xsl:param name="prix"/>
    <xsl:value-of select="format-number($prix,'##,##0.00 E   ')"/>
    <xsl:value-of select="format-number($prix*$tauxFF,'##,##0.00 FF   ')"/>
    <xsl:value-of select="format-number($prix*$tauxATS,'##,##0.00 ATS   ')"/>
  </xsl:template>
</xsl:stylesheet>
```

Le programme *Facture.xsl* engendre la facture *Facture.txt* à partir du document *Articles.xml*. Il affiche pour chaque article son nom suivi de son prix en Euros, Francs et Schillings. La transformation et l'affichage des prix sont effectués par la dernière règle nommée `Conversion`.

Exemple .43 *Facture.txt* : *Facture en Euros, FF et ATS.*

```
PC Portable    1,000.00 E   6,559.57 FF   13,760.30 ATS
Lecteur DVD    767.60 E   5,035.13 FF   10,562.41 ATS
Remise  5%     -88.38 E   -579.73 FF   -1,216.14 ATS

Somme          1,679.22 E   11,014.96 FF   23,106.57 ATS
```

Afin de pouvoir utiliser des caractères spéciaux dans le préfixe/suffixe d'une chaîne de formattage, les caractères spéciaux du tableau 4 peuvent être redéfinis par l'élément du premier niveau `xsl:decimal-format` (page **11**). Voici un exemple.

Exemple .44 *FactureDec.xsl* : *Programme de transformation en FF, ATS et DM*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:import href="Facture.xsl"/>

  <xsl:output method="text" encoding="ISO-8859-1"/>

  <xsl:decimal-format decimal-separator="," grouping-separator="." />
  <xsl:decimal-format name="alpha" zero-digit="a" />

  <xsl:template name="Conversion">
    <xsl:param name="prix"/>
    <xsl:value-of select="format-number($prix,' 00.000,00 E;-00.000,00 E')"/>
    <xsl:value-of select="format-number(position()-1,' (a)','alpha')"/>
  </xsl:template>

</xsl:stylesheet>
```

Le programme *FactureDec.xsl* réutilise les règles du programme *Facture.xsl* et définit deux formats avec l'élément `xsl:decimal-format` qui sont ensuite appliqués dans une nouvelle définition de la règle `Conversion` qui n'affiche que le prix en Euros : l'instruction `xsl:decimal-format` intervertit l'utilisation des caractères « , » et « . » comme séparateurs de groupes et de la partie entière d'un nombre. Le deuxième format indique un compteur alphanumérique qui est utilisé pour numéroter les lignes de facture avec les caractères a, b, c etc. On obtient le résultat suivant :

Exemple .45 *FactureDec.txt* : *Facture en Euros, FF et ATS*

```
PC Portable     01.000,00 E (a)
Lecteur DVD     00.767,60 E (b)
Remise  5%     -00.088,38 E (a)

Somme           01.679,22 E (a)
```

function-available

`Boolean` *`function-available`* `(string` *`nomFonction`*`)`

Cette fonction teste si une fonction XSLT est connu due processeur. Elle peut être utile pour vérifier l'existence de fonctions fournies seulement par certains processeurs, ou apparues dans de nouvelles versions de XSLT.

generate-id

`string` *`generate-id`* `(node-set` *`nœud`*`)`

Cette fonction engendre un identifiant unique pour le nœud passé en argument. S'il y a plusieurs nœuds, seul le premier (dans l'ordre du document) est pris en compte.

Au cours d'une transformation, le processeur XSLT doit *toujours* retourner la même valeur quand cette fonction est appelée pour un même nœud. On dispose ainsi d'un moyen sûr de créer un index unique sur l'ensemble des nœuds du document source. En revanche il n'existe aucune garantie que deux transformations successives engendreront les mêmes identifiants pour les mêmes nœuds.

Cette fonction est principalement utile pour gérer des références à des nœuds indépendamment de leur contenu. Voir page **??** pour une illustration.

id

`node-set` *`id`* `(object` *`arg`*`)`

Cette fonction permet de rechercher un ou plusieurs éléments dans un document par leur `id`. Il est essentiel que ce document soit associé à une DTD qui déclare quels sont les attributs qui identifient les nœuds du document.

D'une manière générale, la fonction prend en entrée un ensemble de nœuds, extrait pour chacun la valeur textuelle et s'en sert comme clé de recherche. En pratique on passe le plus souvent une chaîne de caractères correspondant à l'`id` de l'objet recherché.

Exemple

Voici un document contenant des identifiants et références à identifiants.

Exemple .46 *FilmsLiens.xml* : *Un document avec identifiants*

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE FILMS SYSTEM "FILMS.dtd">

<FILMS>
<FILM>
  <TITRE>Impitoyable</TITRE>
  <ANNEE>1992</ANNEE>
  <MES idref="20"></MES>
</FILM>
<FILM>
  <ANNEE>1995</ANNEE>
  <TITRE>Seven</TITRE>
  <MES idref="31"></MES>
</FILM>

<FILM>
  <ANNEE>1997</ANNEE>
  <TITRE>Les pleins pouvoirs</TITRE>
  <MES idref="20"></MES>
</FILM>

<ARTISTE id="20">
  <NOM>Eastwood</NOM>
  <PNOM>Clint</PNOM>
  <ANNEENAISS>1930</ANNEENAISS>
</ARTISTE>
<ARTISTE id="31">
  <NOM>Fincher</NOM>
  <PNOM>David</PNOM>
  <ANNEENAISS>1962</ANNEENAISS>
</ARTISTE>
</FILMS>
```

La DTD *FILMS.dtd* doit explicitement indiquer que l'attribut `id` de <`ARTISTE`> est un identifiant, et l'attribut `idref` de `FILM` une référence à un identifiant. On doit donc trouver les déclarations suivantes (voir chapitre **??**, tableau **??**, page **??**).

```
<!ATTLIST ARTISTE id ID #REQUIRED>
<!ATTLIST MES idref IDREF #REQUIRED>
```

Voici le programme XSLT qui remplace la référence au metteur en scène par le nom dans les éléments <`FILM`>.

Exemple .47 *Id.xsl* : *La fonction id()*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"
         encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <RESULTAT>
     <xsl:apply-templates select="FILMS/FILM"/>
    </RESULTAT>
  </xsl:template>

  <xsl:template match="FILM">
    <FILM>
     <TITRE><xsl:value-of select="TITRE"/></TITRE>
     <MES><xsl:value-of select="id(MES/@idref)/NOM"/></MES>
    </FILM>
  </xsl:template>
</xsl:stylesheet>
```

Remarquons que *id*() renvoie un ensemble de nœuds, et qu'il est donc possible de l'intégrer dans une expression XPath comme, ici, `id(MES/@idref)/NOM`. On obtient le résultat suivant :

Exemple .48 *Id.xml* : *Résultat obtenu*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<RESULTAT>
<FILM>
<TITRE>Impitoyable</TITRE>
<MES>Eastwood</MES>
</FILM>
<FILM>
<TITRE>Seven</TITRE>
<MES>Fincher</MES>
</FILM>
<FILM>
<TITRE>Les pleins pouvoirs</TITRE>
<MES>Eastwood</MES>
</FILM>
</RESULTAT>
```

L'utilisation des clés (éléments `xsl:key` et fonction*key*()) est plus générale puisqu'elle n'impose ni la présence au préalable de l'information sur les liens dans le document XML, ni la référence à une DTD. Voir page **??**.

key

`node-set` *`key`* `(string` *`nomClé`*`, object` *`expression`*`)`

La fonction *key*() est associée à l'élément `xsl:key` décrit page **15**. Rappelons que cet élément définit un groupe de nœud, lui attribue un nom et définit l'expression (la clé) qui permet de différencier les nœuds au sein de ce groupe.

La fonction prend deux arguments : le nom de la clé, et la valeur de la clé. Elle renvoie les nœuds identifiés par cette valeur.

Exemples

Voir page **15** pour l'élément `xsl:key`, et page **??** pour l'utilisation de cette fonctionnalité.

lang

`Boolean` *`lang`* `(string` *`codeLangue`*`)`

Cette fonction vérifie que la langue du nœud contexte est *`codeLangue`*, et renvoie `true` ou `false` selon le cas. Rappelons que la langue d'un élément est défini par l'attribut réservé `xml :lang`, soit dans l'élément lui-même, soit dans le plus proche ancêtre ayant cet attribut.

last

`number` *`last`* `()`

La fonction *last*() renvoie la taille du contexte XPath. Dans XSLT il s'agit du nombre de nœuds sélectionnés par l'expression de l'attribut `select` dans `xsl:apply-templates` ou `xsl:for-each`. Si la fonction est utilisée dans un prédicat XPath, elle correspond au nombre de nœuds sélectionnés par l'étape précédente.

Il faut être attentif aux nœuds textes vides qui peuvent être comptés ou non selon que l'instruction `xsl:strip-space` a été utilisée ou pas.

local-name

`string` *`local-name`* `(node-set` *`nœud`*`?)`

La fonction renvoie le nom local (sans le préfixe correspondant à l'espace de nom) du nœud passé en argument. Si l'argument est un ensemble, le premier nœud est pris en compte. Si l'argument est omis la fonction s'applique au nœud contexte.

name

`string` *`name`* `(node-set` *`nœud`*`?)`

La fonction renvoie le nom complet, comprenant éventuellement le préfixe correspondant à l'espace de nom, du nœud passé en argument. Si l'argument est un ensemble, le premier nœud est pris en compte. Si l'argument est omis la fonction s'applique au nœud contexte.

namespace-uri

`string` *`namespace-uri`* `(node-set` *`nœud`*`?)`

La fonction renvoie l'URI de l'espace de noms associé au préfixe du nœud passé en argument. La chaîne est vide si le nœud n'appartient à aucun espace de noms.

normalize-space

`string` *`normalize-space`* `(string` *`chaîne`*`?)`

Cette fonction « normalise » la chaîne de caractères passée en argument, ou le contenu textuel du nœud courant si aucun argument n'est passé. La normalisation consiste à supprimer tous les espaces (à savoir, caractères blancs, tabulations, retours chariot et sauts de lignes) en début et en fin de chaîne, et à réduire les séquences de plusieurs espaces consécutifs à un seul.

Cette fonction est décrite page **??**, et intervient dans le contexte plus général de la gestion des espaces avec XSLT (voir page **??**).

not

`Boolean` *`not`* `(Boolean` *`valeur`*`?)`

Cette fonction calcule la négation de son argument.

number

`number` *`number`* `(object` *`valeur`*`?)`

Cette fonction convertit l'argument valeur en numérique. Voir des exemples dans le chapitre **??**, page **??**.

position

`number` *`position`* `()`

Cette fonction renvoie la position du nœud courant dans l'ensemble de nœuds constituant le contexte XPath.

round

`number` *`round`* `(number` *`valeur`*`)`

Cette fonction arrondit son argument à l'entier le plus proche.

starts-with

`Boolean` *`starts-with`* `(string` *`chaîne`*`, string` *`sous-chaîne`*`)`

Cette fonction teste si *`chaîne`* débute avec *`sous-chaîne`*.

string

`string` *`string`* `(object` *`valeur`*`?)`

Cette fonction transforme son argument en chaîne de caractères. Les règles de transformation et des exemples sont décrits dans le chapitre **??**, page **??**. L'appel de cet fonction sans argument s'applique à la valeur textuelle du nœud contexte.

string-length

`number` *`string-length`* `(string` *`chaîne`*`?)`

Cette fonction retourne la longueur de la chaîne de caractères passée comme argument. L'appel de cet fonction sans argument s'applique à la valeur textuelle du nœud contexte.

substring

`string` *`substring`* `(string` *`chaîne`*`, number` *`début`*`, number` *`longueur`*`?)`

Cette fonction permet d'extraire une sous-chaîne de son premier paramètre. Le début et la longueur de cette chaîne sont définis par les paramètres *`début`* et *`longueur`*. Par exemple, l'appel `substring('163.173.212.26', 9, 3)` retourne la chaîne de caractères `'212'`.

La spécification de la longueur peut être omise et se traduit par une extraction à partir de la position jusqu'à la fin de la chaîne passée comme argument. L'appel `substring('163.173.212.26', 9)` retourne la chaîne de caractères `'212.26'`.

substring-after

`string` *`substring-after`* `(string` *`chaîne`*`, string` *`sous-chaîne`*`)`

Cette fonction retourne la chaîne de caractères après la première occurrence de *`sous-chaîne`* dans *`chaîne`*. Par exemple, `substring-after('Jules et Jim', ' et ')` retourne le mot `'Jim'`.

Cette fonction et la fonction *substring-before*() sont surtout utiles pour décomposer une séquence de mots séparés par des blancs. Voir chapitre **??**, page **??** pour un exemple.

substring-before

`string` *`substring-before`* `(string` *`chaîne`*`, string` *`sous-chaîne`*`)`

Cette fonction retourne la chaîne de caractères après la première occurrence de *`sous-chaîne`* dans *`chaîne`*. Par exemple, `substring-before('Jules et Jim', ' et ')` retourne le mot `'Jules'`.

Cette fonction et la fonction *substring-after*() sont surtout utiles pour décomposer une séquence de mots séparés par des blancs. Voir chapitre **??**, page **??**, pour un exemple.

sum

`number` *`sum`* `(node-set` *`nœuds`*`)`

Cette fonction calcule la somme des valeurs numériques des nœuds dans le `node-set` passé en argument. Voir la fonction *format-number*(), page **39**, pour un exemple.

system-property

`object` *`system-property`* `(string` *`nomPropriété`*`)`

Cette fonction retourne la valeur de la propriété système *`nomPropriété`*. La recommandation XSLT définit trois propriétés qui doivent être fournies par tous les processeurs XSLT :

1. `xsl :version` : numéro de la version XSLT implantée par le processeur (`1.0`, `1.1`, ou `2.0`).

2. `xsl :vendor` : nom du vendeur/fournisseur du processeur XSLT. Par exemple la valeur pour le processeur Xalan est `Apache Software Foundation`.
3. `xsl :vendor-uri` : URL du site Web du fournisseur. Pour le processeur Xalan cette propriété prend comme valeur la chaîne `'http ://xml.apache.org/xalan'`.

D'autres propriétés systèmes peuvent être fournies. Elles font généralement partie d'un espace de noms différent de l'espace de nom XSLT.

translate

`string` *`translate`* `(string` *`chaîne`*`, string` *`arg1`*`, string` *`arg2`*`)`

Cette fonction permet de remplacer ou effacer des caractères dans la chaîne de caractères *`chaîne`*. Le principe est de spécifier d'abord dans le paramètre *`arg1`* les caractères à remplacer ou à effacer. Le paramètre *`arg2`* permet ensuite de décider si un caractère est effacé ou remplacé :

- di le paramètre *`arg2`* est vide tous les caractères désignés dans la chaîne *`arg1`* seront effacés dans *`chaîne`* ;
- sinon, avant d'effacer un caractère x, on trouve sa position dans la chaîne *`arg1`* et on vérifie si *à la même position* il existe un caractère y dans la chaîne *`arg2`* : dans ce cas, on remplace x par y.

Par exemple `translate('(+)33-01-4027-2458','-()+',")` efface tous les caractères sauf les numéros dans la chaîne de caractère et retourne `'330140272458'`. En ajoutant un blanc dans le dernier paramètre de la fonction, le premier caractère dans `'-()+'` ne sera pas effacé, mais remplacé par un blanc. Pour tous les autres caractères dans `'-()+'` il n'existe pas de remplaçant à la même position dans la chaîne `' '`, et ils seront effacés. Ainsi, `translate('(+)33-01-4027-2458','-()+',' ')` retourne `'33 01 4027 2458'`.

true

`Boolean` *`true`* `()`

Cette fonction renvoie `true`, et peut être utilisée pour palier l'absence de constantes booléennes dans XSLT. En pratique les conversions effectuées automatiquement sont le plus souvent suffisantes, et évitent d'avoir à écrire « *`expression=true()`* » alors que « *`expression`* » a une signification équivalente.

unparsed-entity-uri

`string` *`unparsed-entity-uri`* `(string` *`nom`*`)`

Cette fonction donne accès aux entités non-XML. Plus précisément, elle retourne l'URL de l'entité référencé. Voir exemple page **??**.

Index