Le langage XSL

Le langage XSL (pour *XML Stylesheet Language*) a été conçu pour transformer des documents XML en d'autres formats comme PDF ou des pages HTML. Au cours de son développement, le projet s'est avéré plus complexe que prévu et il a été scindé en deux unités distinctes XSLT et XSL-FO. Le langage XSLT (pour *XML Stylesheet Language Transformation*) est un langage de transformation de documents XML. Le langageXSL-FO (pour *XML Stylesheet Language - Formatting Objets*) est un langage de mise en page de document. Le processus de transformation d'un document XML en un document imprimable, au format PDF par exemple, est donc découpé en deux phases. Dans la première phase, le document XML est transformé en un document XSL-FO à l'aide de feuilles de style XSLT. Dans la seconde phase, le document FO obtenu à la première phase est converti par un processeur FO en un document imprimable.

Même si le langage XSLT puise son origine dans la transformation de documents XML en document XSL-FO, il est adapté à la transformation d'un document de n'importe quel dialecte XML dans un document de n'importe quel autre dialecte XML. Il est souvent utilisé pour produire des documents XSL-FO ou XHTML, il peut aussi produire des documents SVG.

Ce chapitre est consacré à la partie XSLT de XSL. Le cours est essentiellement basé sur des exemples. Les différentes constructions du langage XSLT sont illustrées pas des fragments de programmes extraits desexemples.

8.1. Principe

Le principe de fonctionnent de XSLT est le suivant. Une feuille de style XSLT contient des règles qui décrivent des transformations. Ces règles sont appliquées à un document source XML pour obtenir un nouveau document XML résultat. Cette transformation est réalisée par un programme appelé *processeur XSLT*. La feuille de style est aussi appelée *programme* dans la mesure où il s'agit des instructions à exécuter par le processeur.

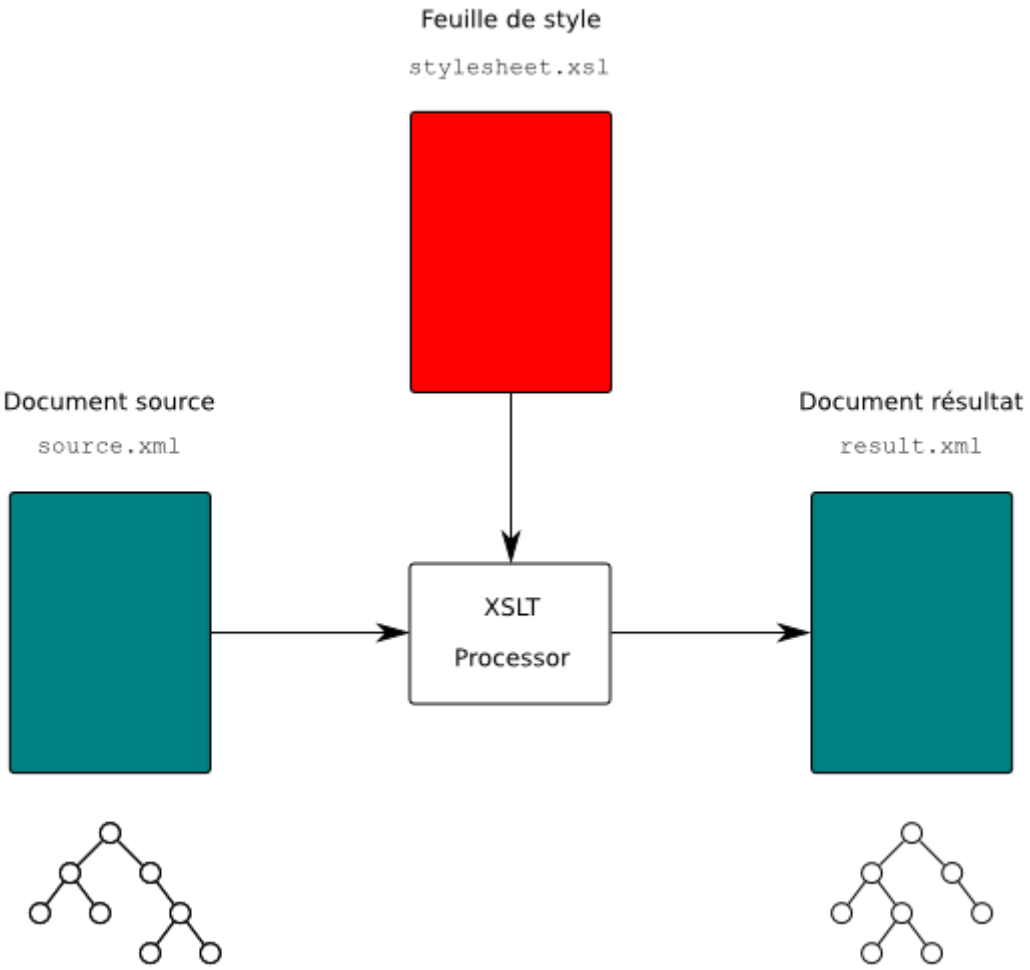

Figure 8.1. Principe de XSLT

La version 2.0 de XSLT a introduit un certain nombre d'évolutions par rapport à la version 1.0. La première évolution est l'utilisation de XPath 2.0 à la place de XPath 1.0. La seconde évolution importante est la possibilité de traiter un document validé au préalable par un schéma XML.

L'intérêt de cette validation est d'associer un type à chaque contenu d'élément et et à chaque valeur d'attribut. Si le type d'un attribut est, par exemple, `xsd:integer` et que sa valeur `123`, celle-ci est interprétée comme un entier et non comme une chaîne de caractères à la manière de XSLT 1.0. La validation par un schéma n'est pas nécessaire pour utiliser XSLT 2.0. Il existe donc deux façons de traiter un document avec XSLT 2.0, soit sans schéma soit avec schéma. La première façon correspond au fonctionnement de XSLT 1.0. La seconde façon prend en compte les types associés aux nœuds par la validation.

Pour la version 1.0 de XSLT, il existe plusieurs processeurs libres dont le plus répandu est `xsltproc`. Il est très souvent déjà installé sur les machines car il fait partie de la librairie standard `libxslt`. En revanche, il n'implémente que la version 1.0 de la norme avec quelques extensions. Le logiciel `saxon` implémente la XSLT 2.0 mais la version gratuite n'implémente que le traitement sans schéma. Il n'existe pas actuellement de processeur libre implémentant le traitement avec schéma. Pour cette raison, ce chapitre se concentre sur le traitement sans schéma même si l'essentiel reste encore valable dans le traitement avec schéma.

Le langage XSLT est un dialecte XML. Ceci signifie qu'une feuille de style XSLT est un document XML qui peut lui-même être manipulé ou produit par d'autres feuilles de style. Cette possibilité est d'ailleurs exploité par schematron pour réaliser une validation en plusieurs phases.

8.2. Premier programme : `Hello, World!`

On commence par la feuille de style XSLT la plus simple. Cette-ci se contente de produire un document XHTML affichant le message `Hello World!` dans un titre. Cette feuille de style a donc la particularité de produire un document indépendant du document source.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"❶
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"❷
                xmlns="http://www.w3.org/1999/xhtml">❸
  <xsl:template match="/">❹
    <html>❺
      <head>
        <title>Hello World!</title>
      </head>
      <body>
        <h1>Hello world!</h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

❶ Élément racine `xsl:stylesheet` de la feuille de style.
❷ Déclaration de l'espace de noms XSLT associé au préfixe `xsl`.
❸ Déclaration de l'espace de noms XHTML comme espace de noms par défaut.
❹ Définition d'une règle s'appliquant à la racine `'/'` du document source.

❺ Fragment de document XHTML retourné par la règle.

Cette feuille de style est constituée d'une seule règle introduite par l'élement `xsl:template` dont l'attribut `match` précise que cette règle s'applique à la racine du document source. L'élément `xsl:template` contient le document XHTML produit. En appliquant ce programme à n'importe quel document XML, on obtient le résultat suivant qui est un document XHTML valide.

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello world!</h1>
  </body>
</html>
```

L'entête XML du résultat à été automatiquement mise par le processeur XSLT. Comme le codage du document résultat n'est pas spécifié par la feuille de style, c'est le codage par défaut de XML qui a été choisi. Le processeur a inséré la déclaration de l'espace de noms XHTML dans l'élément racine `html` du document. Le processeur a également ajouté l'élément `meta` propre à HTML pour préciser le codage des caractères.

8.3. Modèle de traitement

Le document XML source est transformé en un document XML résultat obtenu en appliquant les règles de la *feuille de style* à des nœuds du document source. Chaque application de règle à un nœud produit un fragment de document XML. Tous ces fragments sont assemblés pour former le document résultat.

Chaque fragment produit par l'application d'une règle est une suite de nœuds représentant des éléments, des attributs, des instructions de traitement et des commentaires. Il s'agit le plus souvent d'une suite d'éléments ou d'attributs. Lors de l'assemblage des fragments, ces nœuds viennent s'insérer à l'intérieur d'un autre fragment.

Chaque règle est déclarée par un élément `xsl:template`. Le contenu de cet élément est le fragment de document qui est produit par l'application de cette règle. Ce contenu contient des éléments de l'espace de noms XSLT et des éléments d'autres espaces de noms. Ces derniers éléments sont recopiés à l'identique pour former le

fragment. Les éléments de XSLT sont des instructions qui sont exécutées par le processeur XSLT. Ces éléments sont remplacés dans le fragment par le résultat de leur exécution. L'élément essentiel est l'élément `xsl:apply-templates` qui permet d'invoquer l'application d'autres règles. Les fragments de document produits par ces applications de règles remplacent l'élément `xsl:apply-templates`. L'endroit où se trouve l'élément `xsl:apply-templates` constitue donc le point d'ancrage pour l'insertion du ou des fragments produits par son exécution.

La forme globale d'une règle est donc la suivante.

```
<xsl:template match="...">
  <!-- Fragment produit -->
  ...
  <!-- Application de règles -->
  <xsl:apply-templates .... />
  ...
  <!-- Application de règles -->
  <xsl:apply-templates .... />
  ...
<xsl:template/>
```

Chacune des règles est déclarée avec un élément `xsl:template` dont l'attribut `match` précise sur quels nœuds elle est susceptible d'être appliquée. Le processus de transformation consiste à appliquer des règles sur des nœuds *actifs* du documents source. Au départ, seule la racine est active et la première règle est donc appliquée à cette racine. L'application de chaque règle produit un fragment de document qui va constituer une partie du document résultat. Elle active d'autres nœuds avec des éléments `xsl:apply-templates` placés au sein du fragment de document. Des règles sont alors appliquées à ces nouveaux nœuds actifs. D'une part, elles produisent des fragments de documents qui s'insèrent dans le document résultat à la place des éléments `xsl:apply-templates` qui les ont provoquées. D'autre part, elles activent éventuellement d'autres nœuds pour continuer le processus. Ce dernier s'arrête lorsqu'il n'y a plus de nœuds actifs.

Le processus de transformation s'apparente donc à un parcours de l'arbre du document source. Il y a cependant une différence importante. Dans un parcours classique d'un arbre comme les parcours en largeur ou en profondeur, le traitement d'un nœud entraîne le traitement de ses enfants. L'application d'une règle XSLT active d'autres nœuds mais ceux-ci ne sont pas nécessairement les enfants du nœud sur lequel la règle s'applique. Les nœuds activés sont déterminés par l'attribut `select` des éléments `xsl:apply-templates`. Chaque attribut `select` contient une expression XPath dont l'évaluation donne la liste des nœuds activés.

La figure ci-dessous illustre la construction de l'arbre résultat par l'application des règles XSLT. Chacun des triangles marqués `template` représente un fragment de document produit par l'application d'une règle. Tous ces triangles sont de même taille sur la figure même si les fragments ne sont pas identiques. Les flèches marquées `apply-templates` symbolisent l'application de nouvelles règles par l'activation de nœuds. L'arbre du document résultat est, en quelque sorte, obtenu en contractant ces flèches marquées `apply-templates` et en insérant à leur point de départ le triangle sur lequel elles pointent. Les flèches partant d'un même triangle peuvent partir de points différents car un même fragment de document peut contenir plusieurs éléments `xsl:apply-templates`.

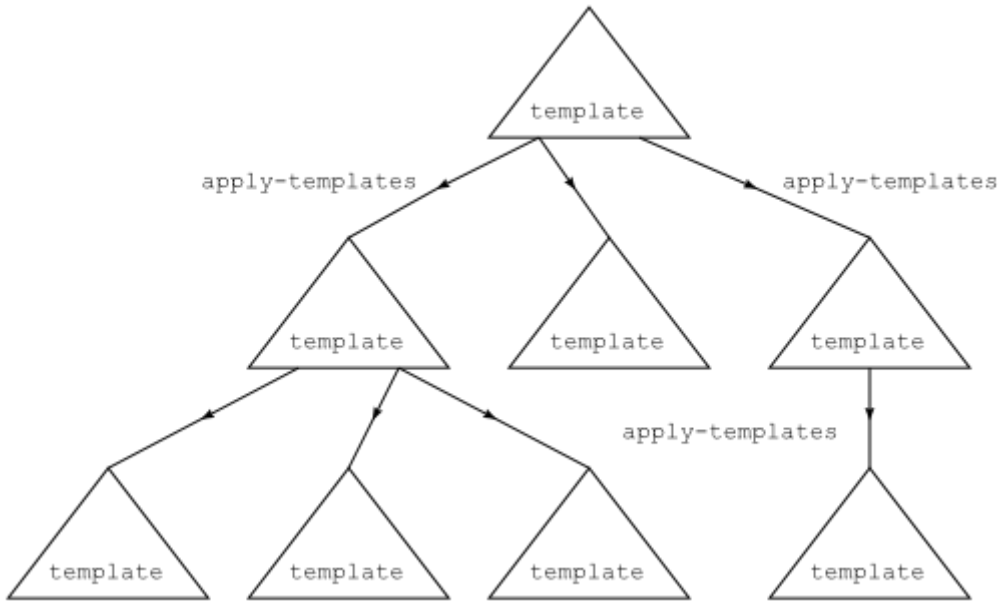

Figure 8.2. Traitement

Il est maintenant possible de revenir sur le premier programme `Hello, Word!` et d'en expliquer le fonctionnement. Ce programme contient une seule règle qui s'applique à la racine du document source. Comme le premier nœud actif au départ est justement la racine, le processus commence par appliquer cette règle. Le document résultat est construit en ajoutant à sa racine le contenu de la règle. Comme ce contenu ne contient aucun élément`xsl:apply-templates`, aucun nouveau nœud n'est rendu actif et le processus de transformation s'arrête après l'application de cette première règle.

8.4. Entête

Le programme ou *feuille de style* est entièrement inclus dans un élément `xsl:stylesheet` ou de façon complètement équivalente un

élément `xsl:transform`. L'attribut `version` précise la version de XSLT utilisée. Les valeurs possibles sont `1.0` ou `2.0`. Un processeur XSLT 1.0 signale généralement une erreur lorsque la feuille de style n'utilise pas cette version. Un processeur XSLT 2.0 passe dans un mode de compatibilité avec la version 1.0 lorsqu'il rencontre une feuille de style de XSLT 1.0. L'espace de noms des éléments de XSLT doit être déclaré. Il est identifié par l'URI `http://www.w3.org/1999/XSL/Transform`. Le préfixe `xsl` est généralement associé à cet espace de noms. Dans tout ce chapitre, ce préfixe est utilisé pour qualifier les éléments XSLT. Il est aussi indispensable de déclarer les espaces de noms du document résultat si celui-ci en utilise. Ces déclarations d'espaces de noms sont importantes car le contenu de chaque règle contient un mélange d'éléments XSLT et d'éléments du document résultat.

Les processeurs XSLT ajoutent les déclarations nécessaires d'espaces de noms dans le document résultat. Il arrive que certains espaces de noms soient déclarés alors qu'ils ne sont pas indispensables. L'attribut `exclude-result-prefixes` de `xsl:stylesheet` permet d'indiquer que certains espaces de noms ne seront pas utilisés dans le document résultat et que leurs déclarations doivent être omises. Cet attribut contient une liste de préfixes séparés par des espaces. Dans l'exemple suivant, les espaces de noms associés aux préfixes `xsl` (XSLT) et `dbk` (DocBook) ne sont pas déclarés dans le document résultat.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
                exclude-result-prefixes="xsl dbk"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dbk="http://docbook.org/ns/docbook"
                xmlns="http://www.w3.org/1999/xhtml">
  ...
```

L'élément `xsl:output` doit être un enfant de l'élément `xsl:stylesheet`. Il permet de contrôler le format du document résultat. Son attribut `method` qui peut prendre les valeurs `xml`, `xhtml`, `html` et `text` indique le type de document résultat produit. Ses attributs `encoding`, `doctype-public`, `doctype-system` précisent respectivement l'encodage du document, le FPI et l'URL de la DTD. Un exemple typique d'utilisation est le suivant.

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<xsl:output method="xml"
            encoding="iso-8859-1"
```

```
            doctype-public="-//W3C//DTD XHTML 1.1//EN"
            doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
            indent="yes"/>
```

8.4.1. Traitement des espaces

Avant d'appliquer la feuille de style au document source, est effectué un traitement préalable des caractères d'espacement dans le document. Ce traitement consiste à supprimer certains nœuds textuels ne contenant que des caractères d'espacement. Seuls les nœuds textuels contenant uniquement des caractères d'espacement sont concernés. Les nœuds contenant, au moins, un autre caractère ne sont jamais supprimés par ce traitement. L'attribut `xml:space` ainsi que les éléments `xsl:strip-space` et `xsl:preserve-space` contrôlent lesquels des nœuds textuels sont effectivement supprimés.

La sélection des nœuds textuels à supprimer est basée sur une liste de noms d'éléments du document source dont les caractères d'espacement doivent être préservés. Par défaut, cette liste contient tous les noms d'éléments et aucun nœud textuel contenant des caractères d'espacement n'est supprimé. L'élément `xsl:strip-space` permet de retirer des noms d'éléments de cette liste et l'élément `xsl:preserve-space` permet d'en ajouter. Ce dernier élément est rarement utilisé puisque la liste contient, au départ, tous les noms d'éléments. L'attribut `elements` des éléments `xsl:strip-space` et `xsl:preserve-space` contient une liste de noms d'éléments (à retirer ou à ajouter) séparés par des espaces. Cet attribut peut également contenir la valeur `'*'` ou des valeurs de la forme `tns:*`. Dans ces cas, sont retirés (pour `xsl:strip-space`) ou ajoutés (pour `xsl:preserve-space`) à la liste tous les noms d'éléments ou tous les noms d'éléments de l'espace de noms associé au préfixe `tns`.

Un nœud textuel est retiré de l'arbre du document si les deux conditions suivantes sont vérifiées. Il faut d'abord que le nom de son parent n'appartienne pas à la liste des noms d'éléments dont les espaces doivent être préservés. Ceci signifie que le nom de son parent a été retiré de la liste grâce à un élément `xsl:strip-space`. Il faut ensuite que la valeur de l'attribut `xml:space` donné au plus proche parent contenant cet attribut ne soit pas la valeur `preserve`. La valeur par défaut de cet attribut est `default` qui autorise la suppression.

La feuille de style suivante recopie le document source mais les nœuds textuels contenant uniquement des caractères d'espacement sont supprimés du contenu des éléments `strip`. Afin d'observer les effets de `xsl:strip-space`, il est nécessaire que la valeur de l'attribut `indent` de l'élément `xsl:output` soit `no` pour qu'aucun caractère d'espacement ne soit ajouté pour l'indentation.

```
<?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" indent="no"/>
  <xsl:strip-space elements="strip"/>
  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Le document suivant contient des éléments `preserve` et des éléments `strip`.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<list>
  <sublist>
    <preserve>
      <p>a b c</p>   <p> a b c </p>
    </preserve>
    <strip xml:space="preserve">
      <p>a b c</p>   <p> a b c </p>
    </strip>
    <strip>
      <p>a b c</p>   <p> a b c </p>
    </strip>
  </sublist>
  <sublist xml:space="preserve">
    <strip>
      <p>a b c</p>   <p> a b c </p>
    </strip>
    <strip xml:space="default">
      <p>a b c</p>   <p> a b c </p>
    </strip>
  </sublist>
</list>
```

Si la feuille de style précédente est appliquée au document précédent, certains espaces sont supprimés. Aucun des espaces contenus dans les éléments `p` n'est

supprimé car ces éléments ont un seul nœud textuel comme enfant et ce nœud textuel contient des caractères autres que des espaces. En revanche, des retours à la ligne entre les balises `<strip>` et `<p>` ainsi que les espaces entre les balises `</p>` et `<p>` disparaissent car les nœuds textuels qui les contiennent ne contiennent que des caractères d'espacement.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<list>
  <sublist>
    <preserve>
      <p>a b c</p>   <p> a b c </p>
    </preserve>
    <!-- La valeur "preserve" de l'attribut xml:space
         inhibe la suppression des espaces -->
    <strip xml:space="preserve">
      <p>a b c</p>   <p> a b c </p>
    </strip>
    <strip><p>a b c</p><p> a b c </p></strip>
  </sublist>
  <sublist xml:space="preserve">
    <!-- La valeur "preserve" de l'attribut xml:space du parent
         inhibe la suppression des espaces -->
    <strip>
      <p>a b c</p>   <p> a b c </p>
    </strip>
    <!-- La valeur "default" de l'attribut xml:space masque
         la valeur "preserve" de l'attribut du parent -->
    <strip xml:space="default"><p>a b c</p><p> a b c </p></strip>
  </sublist>
</list>
```

8.5. Définition et application de règles

Les deux éléments qui constituent le cœur de XSLT sont les éléments `xsl:template` et `xsl:apply-templates` qui permettent respectivement de définir des règles et de les appliquer à des nœuds. Chaque définition de règle par un élément `xsl:template` précise, par un motif XPath, les nœuds sur lesquels elle s'applique. Chaque application de règles par un élément `xsl:apply-templates` précise les nœuds actifs auxquels doit être appliquée une règle mais ne

spécifie pas du tout la règle à appliquer. Le choix de la règle à appliquer à chaque nœud est fait par le processeur XSLT en fonction de priorités entre les règles. Ces priorités sont déterminées à partir des motifs XPath de chaque règle. Ce principe de fonctionnement est le mode normal de fonctionnement de XSLT. Il est, cependant, utile à l'occasion d'appliquer une règle déterminée à un nœud. L'élément `xsl:call-template` permet l'appel explicite d'une règle par son nom.

Les définitions de règles sont globales. Tous les éléments `xsl:template` sont enfants de l'élément racine `xsl:stylesheet` et la portée des règles est l'intégralité de la feuille de style. Au contraire, les éléments `xsl:apply-templates` et `xsl:call-template` ne peuvent apparaître que dans le contenu d'un élément `xsl:template`.

8.5.1. Définition de règles

L'élément `xsl:template` permet de définir une règle. Cet élément est nécessairement enfant de l'élément racine `xsl:stylesheet`. L'attribut `match` qui contient un motif XPath définit le contexte de la règle, c'est-à-dire, les nœuds sur lesquels elle s'applique. L'attribut `name` donne le nom de la règle. Un de ces deux attributs doit être présent mais les deux peuvent être simultanément présents. L'attribut `match` permet à la règle d'être invoquée implicitement par un élément `xsl:apply-templates` alors que l'attribut `name` permet à la règle d'être appelée explicitement par un élément `xsl:call-template`. Le contenu de l'élément `xsl:template` est le fragment de document à insérer dans le document résultat lors de l'application de la règle. L'élément `xsl:template` peut aussi contenir un attribut `priority` pour fixer la priorité de la règle ainsi qu'un attribut `mode` pour préciser dans quels modes elle peut être appliquée.

Le fragment de programme suivant définit une règle. La valeur de l'attribut `match` vaut `'/'` et indique donc que la règle s'applique uniquement à la racine de l'arbre. La racine de l'arbre résultat est alors le fragment XHTML contenu dans `xsl:template`. Comme ce fragment ne contient pas d'autres directives XSLT, le traitement de l'arbre source s'arrête et le document résultat est réduit à ce fragment.

```
<xsl:template match="/">
 <html>
   <head>
     <title>Hello, World!</title>
   </head>
   <body>
     <h1>Hello, world!</h1>
   </body>
 </html>
```

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

La règle ci-dessous s'applique à tous les éléments de nom `author`, `year` ou `publisher`.

```
<xsl:template match="author|year|publisher">
  ...
</xsl:template>
```

Lorsqu'une règle est appliquée à un nœud du document source, ce nœud devient le nœud courant du focus. Les expressions XPath sont alors évaluées à partir de ce nœud qui est retourné par l'expression `'.'`. Certains éléments comme `xsl:for-each` ou les filtres peuvent localement changer le focus et le nœud courant. Le nœud sur lequel est appliquée la règle peut encore être recupéré par un appel à la fonction XPath `current()`.

8.5.2. Application implicite

L'élément `xsl:apply-templates` provoque l'application de règles sur les nœuds sélectionnés par son attribut `select` qui contient une expression XPath. L'évaluation de cette expression par rapport au nœud courant donne une liste de nœuds du document source. À chacun de ces nœuds, une règle choisie par le processeur XSLT est appliquée. Le résultat de ces application de règles remplace alors l'élément `xsl:apply-templates`dans le contenu de l'élément `xsl:template` pour former le résultat de la règle en cours d'application.

L'expression XPath de l'attribut `select` sélectionne, en général, plusieurs nœuds. Sur chacun de ces nœuds, est appliquée une seule règle dépendant du nœud. La règle est choisie parmi les règles où il y a concordance entre le motif XPath de l'attribut `match` et le nœud. Lors de l'application la règle choisie, chaque nœud sélectionné devient le nœud courant du focus. Pour appliquer plusieurs règles à un même nœud, il faut plusieurs éléments`xsl:apply-templates`. Il est, en particulier, possible d'avoir recours à des modes pour appliquer des règles différentes.

La valeur par défaut de l'attribut `select` est `node()` qui sélectionne tous les enfants du nœud courant, y compris les nœuds textuels, les commentaires et les instructions de traitement. Pour sélectionner uniquement les enfants qui sont des éléments, il faut mettre la valeur `'*'`. Pour sélectionner tous les attributs du nœud courant, il faut mettre la valeur `@*` qui est l'abréviation de `attribute::*`. Il est, bien sûr, possible de mettre toute expression XPath comme `ancestor-or-self::p[@xml:lang][1]/@xml:lang`. Une partie importante de la programmation XSLT réside dans les choix judicieux des valeurs des attributs `select` des éléments`xsl:apply-templates` ainsi que des valeurs des attributs `match` des éléments `xsl:template`.

Dans la feuille de style suivante, la première règle qui s'applique à la racine `'/'` crée la structure du document XHTML. Les enfants `li` de l'élément `ul` sont créés par les applications de règles sur les éléments `book`provoquées par le premier élément `xsl:apply-templates`. Les contenus des éléments `li` sont construits par les applications de règles sur les enfants des éléments `book` provoquées par le second élément `xsl:apply-templates`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                                version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
    <html>
      <head><title>Bibliographie</title></head>
      <body>
       <h1>Bibliographie</h1>
       <ul><xsl:apply-templates select="bibliography/book"/></ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="book">
    <!-- Une entrée (item) de la liste par livre -->
    <li><xsl:apply-templates select="*"/></li>
  </xsl:template>
  ...
</xsl:stylesheet>
```

Les nœuds sur lesquels sont appliqués les règles sont très souvent des descendants et même des enfants du nœud courant mais ils peuvent également être très éloignés dans le document. Dans l'exemple suivant, les éléments sont retournés par la fonction XPath `id`. Il faut remarquer que l'élément `xsl:for-each` change le focus et que l'expression XPath `'.'` ne désigne plus le nœud courant sur lequel s'applique la règle.

```
<xsl:template match="dbk:co">
  <xsl:text>\includegraphics[scale=0.25]{images/callouts/</xsl:text>
  <xsl:number level="single" count="dbk:co" format="1"/>
  <xsl:text>.pdf}</xsl:text>
</xsl:template>
w<xsl:template match="dbk:callout">
```

```
  <xsl:text>\item[</xsl:text>
  <!-- Parcours des éléments référencés par l'attribut areарefs -->
  <xsl:for-each select="id(@arearefs)">
    <xsl:apply-templates select="."/>
  </xsl:for-each>
  <xsl:text>]</xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

Les nœuds sélectionnés par l'expression XPath de l'attribut `select` sont normalement traités dans l'ordre du document. L'élément `xsl:apply-templates` peut, néanmoins, contenir un ou plusieurs éléments `sort` pour effectuer un tri des nœuds sélectionnés.

8.5.3. Priorités

Lorsque plusieurs règles peuvent s'appliquer à un même nœud, le processeur XSLT choisit la plus appropriée parmi celles-ci. Le choix de la règle est d'abord dicté par les priorités d'import entre les feuilles de style puis par les priorités entre les règles.

Le choix de la règle à appliquer s'effectue en deux étapes. La première étape consiste à ne conserver que les règles de la feuille de style de priorité d'import maximale parmi les règles applicables. Lorsqu'une feuille de style est importée par une autre feuille de style, elle a une priorité d'import inférieure à celle qui l'importe. L'ordre des imports a également une incidence sur les priorités d'import.

La seconde étape consiste à choisir, parmi les règles restantes, celle qui a la priorité maximale. C'est une erreur s'il y en a plusieurs de priorité maximale. Le processeur XSLT peut signaler l'erreur ou continuer en choisissant une des règles. La priorité d'une règle est un nombre décimal. Elle peut est fixée par un attribut `priority` de l'élément `xsl:template`. Sinon, elle prend une valeur par défaut qui dépend de la forme du motif contenu dans l'attribut `match`. Les priorités par défaut prennent des valeurs entre `-0.5` et `0.5`. Lorsque le motif est de la forme *`expr-1`* `|` `...` `|` *`expr-N`*, le processeur considère chacun des motifs *`expr-i`* de façon indépendante. Il fait comme si la règle était répétée pour chacun de ces motifs. L'axe n'a aucune incidence sur la priorité par défaut. Celle-ci est déterminée par les règles ci-dessous.

`-0.5`

pour les motifs très généraux de la forme `*`, `@*`, `node()`, `text()`, `comment()` ainsi que le motif `/`,

`-0.25`

pour les motifs de la forme `*:name` ou `prefix:*` comme `*:p` ou `dbk:*`,

`0`

pour les motifs de la forme `name` ou `@name` comme `section` ou `@type`,

`0.5`

pour tous les autres motifs comme `chapter/section` ou `section[@type]`.

8.5.4. Application de règles moins prioritaires

C'est normalement la règle de priorité maximale qui est appliquée à un nœud lorsque le processus est initié par un élément `xsl:apply-templates`. Il existe également les deux éléments `xsl:next-match` et `xsl:apply-imports` qui permettent d'appliquer une règle de priorité inférieure. Ces deux éléments appliquent implicitement une règle à l'élément courant et il n'ont donc pas d'attribut `select`.

L'élément `xsl:next-match` applique à l'élément courant la règle qui se trouvait juste avant la règle courante dans l'ordre des priorités croissantes. Cet élément est souvent utilisé pour définir une règle pour un cas particulier tout en réutilisant la règle pour le cas général. Dans l'exemple suivant, la règle spécifique pour les éléments `para` avec un attribut `role` égal à `right` ajoute un élément `div` avec un attribut `style`. Elle appelle la règle générale pour la règle pour les éléments DocBook `para`.

```
<!-- Paragraphes -->
<xsl:template match="dbk:para">
  <p xsl:use-attribute-sets="para"><xsl:apply-templates/></p>
</xsl:template>
<!-- Paragraphes alignés à droite -->
<xsl:template match="dbk:para[@role='right']">
  <div style="text-align: right">
    <xsl:next-match/>
  </div>
</xsl:template>
```

L'élément `xsl:apply-imports` permet d'appliquer au nœud courant une règle provenant d'une feuille de style importée par la feuille de style contenant la règle en cours. La règle appliquée est la règle ayant une priorité maximale parmi les règles provenant des feuilles de style importées. Cet élément est souvent utilisé pour redéfinir une règle d'une feuille de style importée tout en utilisant la règle redéfinie. Dans l'exemple suivant, la feuille de style `general.xsl` contient une règle pour les éléments DocBook `para`.

```
<!-- Feuille de style general.xsl -->
<!-- Paragraphes -->
<xsl:template match="dbk:para">
  <p xsl:use-attribute-sets="para"><xsl:apply-templates/></p>
</xsl:template>
```

La feuille de style `main.xsl` importe la feuille de style `general.xsl`. Elle contient une nouvelle règle pour les éléments DocBook `para`. Cette règle a une priorité d'import supérieure et elle est donc appliquée en priorité. Cette règle fait appel à la règle contenue dans `general.xsl` par l'élément `xsl:apply-imports`.

```
<!-- Feuille de style main.xsl -->
<!-- Import de la feuille de style general.xsl -->
<xsl:import href="general.xsl"/>
<!-- Paragraphes alignés à gauche -->
<xsl:template match="dbk:para">
  <div style="text-align: left">
    <xsl:apply-imports/>
  </div>
</xsl:template>
```

8.5.5. Application explicite

L'élément `xsl:call-template` provoque l'application de la règle dont le nom est donné par l'attribut `name`. Contrairement à l'élément `xsl:apply-templates`, le contexte et, en particulier, le nœud courant ne sont pas modifiés pour l'application de la règle nommée. Le résultat de l'application de la règle remplace alors l'élément `xsl:call-template` dans le contenu de l'élément `xsl:template` pour former le résultat de la règle en cours d'application.

L'utilisation de `xsl:call-template` est courant pour factoriser des parties communes à des règles. Lorsque plusieurs règles partagent un fragment, il est approprié de placer ce fragment dans une nouvelle règle nommée puis de l'utiliser en invoquant, à plusieurs reprises, cette règle avec `xsl:call-template`.

Dans la feuille de style suivante, les différentes règles pour traiter les éléments `title`, `url` et les autres enfants de l'élément `book` font appel à la règle `comma` pour ajouter une virgule si l'enfant n'est pas le dernier. Il est important que le focus soit préservé à l'appel de cette règle pour que le test `position() != last()` fonctionne correctement.

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

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

  <xsl:template match="book">
    <!-- Une entrée (item) de la liste par livre -->
    <li><xsl:apply-templates select="*"/></li>
  </xsl:template>
  <xsl:template match="title">
    <!-- Titre du livre en italique -->
    <i><xsl:apply-templates/></i>
    <xsl:call-template name="comma"/>
  </xsl:template>
  <xsl:template match="url">
    <!-- URL du livre en police fixe -->
    <tt><xsl:apply-templates/></tt>
    <xsl:call-template name="comma"/>
  </xsl:template>
  <xsl:template match="*">
    <xsl:apply-templates/>
    <xsl:call-template name="comma"/>
  </xsl:template>
  <xsl:template name="comma">
    <!-- Virgule si ce n'est pas le dernier -->
    <xsl:if test="position() != last()">
      <xsl:text>, </xsl:text>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Les deux éléments `xsl:apply-templates` et `xsl:call-template` peuvent contenir des éléments `xsl:with-param` pour spécifier les valeurs de paramètres que la règle appliquée peut avoir déclarés avec l'élément `xsl:param`.

8.6. Construction de contenu

Chaque application de règle de la feuille de style produit un fragment du résultat. Ce fragment est construit à partir du contenu de l'élément `xsl:template` et d'autres éléments permettant d'insérer d'autres nœuds calculés.

8.6.1. Contenu brut

Lorsqu'une règle est appliquée, tout le contenu de l'élément `xsl:template` est recopié dans le résultat à l'exception des éléments de l'espace de noms XSLT. Ces derniers sont, en effet, évalués par le processeur puis remplacés par leur valeur. Tout les autres éléments ainsi que leur contenu se retrouvent recopiés à l'identique dans le document résultat.

Cette fonctionnalité est utilisée dans le premier exemple `Hello, Word!`. L'unique élément `xsl:template` contient le document XHTML produit par la feuille de style. Les espaces de noms jouent ici un rôle essentiel puisqu'ils permettent de distinguer les directives de traitement pour le processeur XSLT (c'est-à-dire les éléments XSLT) des autres éléments.

```
<xsl:template match="/">
  <html>
    <head>
      <title>Hello World!</title>
    </head>
    <body>
      <h1>Hello world!</h1>
    </body>
  </html>
</xsl:template>
```

L'insertion d'un contenu fixe, comme dans l'exemple précédent, est très limité. Le résultat attendu dépend généralement du document source. Dans la règle ci-dessous, le contenu de l'élément `h1` provient du document source. Ce texte est extrait du document source grâce à l'élément `xsl:value-of` décrit ci-dessous. Il est le contenu textuel de l'élément racine `text` du document source.

```
<xsl:template match="/">
  <html>
    <head>
      <title>Localized Hello World !</title>
    </head>
    <body>
      <h1><xsl:value-of select="text"/></h1>
```

```
    </body>
  </html>
</xsl:template>
```

Si la feuille de style ainsi modifiée est appliquée au document suivant, on obtient un document XHTML où le contenu de l'élément `h1` est maintenant `Bonjour !`.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!-- Élément text contenant le texte inséré dans l'élément h1 -->
<text>Bonjour !</text>
```

Du contenu peut aussi être construit par un élément `xsl:apply-templates`. Ce contenu est alors le résultat de l'application de règles aux éléments sélectionnés par l'attribut `select` de `xsl:apply-templates`. On reprend le document `bibliography.xml` déjà utilisé au chapitre sur syntaxe.

La feuille de style suivante transforme le document `bibliography.xml` en un document XHTML qui présente la bibliographie sous forme d'une liste avec une mise en forme minimaliste Cette feuille de style fonctionne de la manière suivante. La première règle est appliquée à la racine du document source. Elle produit le squelette du document XHTML avec en particulier un élément `ul` pour contenir la liste des livres. Le contenu de cet élément`ul` est obtenu en appliquant une règle à chacun des éléments `book`. La seconde règle de la feuille de style est la règle qui est appliquée aux éléments `book`. Pour chacun de ces éléments, elle produit un élément `li` qui s'insère dans le contenu de l'élément `ul`. Le contenu de l'élément `li` est, à nouveau, produit en appliquant une règle aux enfants de l'élément `book`. C'est la dernière règle qui s'applique à chacun de ces éléments. Cette règle se contente de rappeler récursivement une règle sur leurs enfants. La règle par défaut recopie alors les contenus textuels de ces élément.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<!-- Règle pour la racine qui construit le squelette -->
<xsl:template match="/">
  <html>
    <head>
      <title>Bibliographie</title>
    </head>
    <body>
```

```
      <h1>Bibliographie</h1>
      <ul><xsl:apply-templates select="bibliography/book"/></ul>
    </body>
  </html>
</xsl:template>
<!-- Règle pour les éléments book -->
<xsl:template match="book">
  <!-- Une entrée li de la liste par livre -->
  <li><xsl:apply-templates/></li>
</xsl:template>
<!-- Règle pour les autres éléments -->
<xsl:template match="*">
  <!-- Récupération du texte par la règle par défaut -->
  <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

En appliquant la feuille de syle précédente au document `bibliography.xml`, on obtient le document XHTML suivant dont l'indentation a été remaniée pour une meilleure présentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Bibliographie</title>
  </head>
  <body>
    <h1>Bibliographie</h1>
    <ul>
      <li>XML langage et appplications Alain Michard 2001 Eyrolles
        2-212-09206-7 http://www.editions-eyrolles/livres/michard/ </li>
      <li>Designing with web standards Jeffrey Zeldman 2003 New Riders
        0-7357-1201-8 </li>
      ...
    </ul>
  </body>
</html>
```

Dans la feuille de style précédente, plusieurs règles sont susceptibles de s'appliquer aux éléments `book` : la deuxième règle dont la valeur de l'attribut `match` est `book` et la troisième dont la valeur de l'attribut `match` est `'*'`. Le processeur XSLT choisit la deuxième en raison des priorités attribuées aux règles en fonction du motif contenu dans l'attribut `match`.

La présentation de la bibliographie en XHTML obtenue avec le feuille de style précédente est très sommaire. Il est possible de l'améliorer en mettant, par exemple, le titre en italique et l'URL en fonte fixe. Il suffit, pour cela, d'ajouter deux règles spécifiques pour les éléments `title` et `url`. Les deux nouvelles règles suivantes ont une priorité supérieure à la dernière règle de la feuille de style et elle est donc appliquée aux éléments `title` et `url`.

```
<!-- Règle pour les éléments title -->
<xsl:template match="title">
  <!-- Titre en italique -->
  <i><xsl:apply-templates/></i>
</xsl:template>
<!-- Règle pour les éléments url -->
<xsl:template match="url">
  <!-- URL en fonte fixe -->
  <tt><xsl:apply-templates/></tt>
</xsl:template>
```

Beaucoup de la programmation XSLT s'effectue en jouant sur les éléments sélectionnés par les attributs `match` des éléments `xsl:apply-templates`. La feuille de style suivante présente la bibliographie en XHTML sous la forme de deux listes, une pour les livres en français et une autre pour les livres en anglais. Le changement par rapport à la feuille de style précédente se situe dans la règle appliquée à la racine. Celle-ci contient maintenant deux éléments `xsl:apply-templates`. Le premier active les livres en français et le second les livres en anglais.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
  <xsl:template match="/">
  <html>
    <head>
      <title>Bibliographie Français/Anglais</title>
    </head>
```

```
    <body>
    <h1>Bibliographie en français</h1>
    <!-- Traitement des livres avec l'attribut lang="fr" -->
    <ul><xsl:apply-templates
select="bibliography/book[@lang='fr']"/></ul>
    <h1>Bibliographie en anglais</h1>
    <!-- Traitement des livres avec l'attribut lang="en" -->
    <ul><xsl:apply-templates
select="bibliography/book[@lang='en']"/></ul>
    </body>
  </html>
  </xsl:template>
  <!-- Les autres règles sont inchangées -->
  ...
</xsl:stylesheet>
```

8.6.1.1. Substitution d'espaces de noms

Il est parfois souhaité qu'une feuille de style XSLT produise une autre feuille de style comme document résultat. Ce mécanisme est, en particulier, mis en œuvre par les schematrons. Le problème est que l'espace de noms XSLT est justement utilisé par le processeur pour distinguer les éléments XSLT qui doivent être exécutés des autres éléments qui doivent être recopiés dans le résultat. Il est alors, a priori, impossible de mettre des éléments XSLT dans le document résultat. L'élément `xsl:namespace-alias` permet de contourner cette difficulté. Il provoque la conversion d'un espace de noms de la feuille de style en un autre espace de noms dans le résultat. Cet élément doit être enfant de l'élément racine `xsl:stylesheet` de la feuille de de style. Ses attributs `stylesheet-prefix` et `result-prefix` donnent respectivement les préfixes associés à l'espace de noms à convertir et de l'espace de noms cible. Ces deux attributs peuvent prendre la valeur particulière `#default` pour spécifier l'espace de noms par défaut.

Pour produire des éléments XSLT dans le document résultat, la feuille de style déclare, d'une part, l'espace de noms XSLT associé, comme d'habitude, au préfixe `xsl` et elle déclare, d'autre part, un autre espace de noms associé à un préfixe arbitraire `tns`. Les éléments XSLT devant être recopiés dans le résultat sont placés dans l'espace de noms associé au préfixe `tns`. L'élément `xsl:namespace-alias` assure la conversion de cet autre espace de noms en l'espace de noms XSLT dans le résultat. L'exemple suivant est une feuille de style produisant une autre feuille de style produisant, à son tour, un document XHTML.

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

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:tns="http://www.liafa.jussieu.fr/~carton">
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes"/>
  <!-- Le préfix tns est transformé en xsl dans la sortie -->
  <xsl:namespace-alias stylesheet-prefix="tns" result-prefix="xsl"/>
  <xsl:template match="/">
    <tns:stylesheet version="2.0"
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <tns:output method="xhtml" encoding="iso-8859-1" indent="yes"/>
      <tns:template match="/">
        <html>
          <head>
            <title>Exemple d'expressions XPath</title>
          </head>
          <body>
            <h1>Exemple d'expressions XPath</h1>
            <xsl:apply-templates select="operators/operator"/>
          </body>
        </html>
      </tns:template>
    </tns:stylesheet>
  </xsl:template>
  ...
```

8.6.2. Règles par défaut

Il existe des règles par défaut pour traiter chacun des nœuds possibles d'un document. Celles-ci simplifient l'écriture de feuilles de style très simples en fournissant un traitement adapté à la plupart des nœuds. Ces règles ont la priorité la plus faible. Elles ne s'appliquent à un nœud que si la feuille de style ne contient aucune règle pouvant s'appliquer à ce nœud. L'effet de ces différentes règles est le suivant. Elles suppriment les instructions de traitement et les commentaires. Elles recopient dans le résultat le contenu des nœuds textuels et les valeurs des attributs. Le traitement d'un élément par ces règles est d'appliquer récursivement une règle à ses enfants, ce qui exclut les attributs.

La feuille de style suivante est la feuille de style minimale sans aucune définition de règles. Elle est néanmoins utile grâce à la présence des règles par défaut.

```
<?xml version="1.0" encoding="us-ascii"?>

<!-- Feuille de style minimale sans règle -->

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

En appliquant cette feuille de style minimale au document `bibliography.xml`, on obtient le résultat suivant.

```
<?xml version="1.0" encoding="UTF-8"?>


    XML langage et applications
    Alain Michard
    2001
    Eyrolles
    2-212-09206-7
    http://www.editions-eyrolles/livres/michard/

    XML by Example
    Benoît Marchal
    2000
    Macmillan Computer Publishing
    0-7897-2242-9

    XSLT fondamental
    Philippe Drix
    2002
    Eyrolles
    2-212-11082-0

    Designing with web standards
    Jeffrey Zeldman
    2003
    New Riders
    0-7357-1201-8
```

```

```

Ce résultat s'explique par la présence de règles par défaut qui sont les suivantes.

```
<!-- Règle pour la racine et les éléments -->
<xsl:template match="/|*">
  <!-- Traitement récursif des enfants -->
  <!-- La valeur par défaut de l'attribut select est node() -->
  <xsl:apply-templates/>
</xsl:template>

<!-- Règle pour nœuds textuels et les attributs -->
<xsl:template match="text()|@*">
  <!-- La valeur textuelle est retournée -->
  <xsl:value-of select="."/>
</xsl:template>

<!-- Règle pour les intructions de traitement et les commentaires -->
<!-- Suppression de ceux-ci car cette règle ne retourne rien -->
<xsl:template match="processing-instruction()|comment()"/>
```

La règle par défaut des attributs est d'afficher leur valeur. La valeur des attributs n'apparaissent pas dans le résultat ci-dessus car cette règle par défaut pour les attributs n'est pas invoquée. En effet, la valeur par défaut de l'attribut `select` de l'élément `apply-templates` est `node()` qui ne sélectionne pas les attributs.

8.6.3. Expression XPath en attribut

Il est possible d'insérer directement dans un attribut la valeur d'une expression XPath. L'expression doit être délimitée dans l'attribut par des accolades `'{'` et `'}'`. À l'exécution, l'expression est évaluée et le résultat remplace dans l'attribut l'expression et les accolades qui l'entourent. Un même attribut peut contenir un mélange de texte et de plusieurs expressions XPath comme dans l'exemple `<p ... >` ci-dessous.

```
<body background-color="{$color}">
  ...
  <p style="{$property}: {$value}">
```

Cette syntaxe est beaucoup plus concise que l'utilisation classique de l'élément `xsl:attribute` pour ajouter un attribut.

```
<body>
  <xsl:attribute name="background-color" select="$color"/>
```

Pour insérer des accolades ouvrantes ou fermantes dans la valeur d'un attribut, il faut simplement doubler les accolades et écrire `'{{'` et `'}}'`. Si la règle XSLT suivante

```
<xsl:template match="*">
  <braces id="{@id}" esc="{{@id}}" escid="{{{@id}}}"/>
</xsl:template>
```

est appliquée à un élément `<braces id="JB007"/>`, on obtient l'élément suivant. La chaîne `{@id}` est remplacée par la valeur de l'attribut `id`. En revanche, la chaîne `{{@id}}` donne la valeur `{@id}` où chaque paire d'accolades donne une seule accolade. Finalement, la chaîne `{{{@id}}}` combine les deux comportements.

```
<braces id="JB007" esc="{@id}" escid="{JB007}"/>
```

Les expressions XPath en attribut avec des structures de contrôle XPath sont souvent plus concises que les constructions équivalentes en XSLT.

Les expressions XPath en attribut apparaissent normalement dans les attribut des éléments hors de l'espace de noms XSLT. Ils peuvent égalemement apparaître comme valeur de quelques attributs d'éléments XSLT. C'est, par exemple, le cas de l'attribut `name` des éléments `xsl:element` et `xsl:attribute`.

8.6.4. Texte brut

L'élément `xsl:text` utilise son contenu pour créer un nœud texte dans le document résultat. Les caractères spéciaux `'<'`, `'>'` et `'&'` sont automatiquement remplacés par les entités prédéfinies correspondantes si la valeur de l'attribut `method` de l'élément `output` n'est pas `text`. Si la valeur cet attribut est `text`, les caractères spéciaux sont écrits tels quels.

```
<xsl:text>Texte et entités '&lt;', '&gt;' et '&amp;'</xsl:text>
```

La présentation de la bibliographie en XHTML peut être améliorée par l'ajout de virgules `','` entre les propriétés titre, auteur, … de chacun des livres. L'élément `xsl:text` est alors nécessaire pour insérer les virgules. Il faut, en outre, gérer l'absence de virgule à la fin grâce à l'élément `xsl:if`.

```
<!-- Règle pour les autres éléments -->
<xsl:template match="*">
```

```
  <xsl:apply-templates/>
  <!-- Virgule après ces éléments -->
  <xsl:if test="position() != last()">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:template>
```

Pour que le test `position() != last()` fonctionne correctement, il faut que la règle appliquée aux éléments `book` n'active pas les nœuds textuels contenant des caractères d'espacements. Il faut, pour cela, remplacer la valeur par défaut de l'attribut `select` par la valeur `'*'` qui ne sélectionne que les enfants qui sont des éléments et pas ceux de type `text()`.

```
<!-- Règle pour les éléments book -->
<xsl:template match="book">
  <!-- Une entrée li de la liste par livre -->
  <li><xsl:apply-templates select="*"/></li>
</xsl:template>
```

8.6.5. Expression XPath

L'élément `xsl:value-of` crée un un nœud texte dont le contenu est calculé. L'attribut `select` doit contenir une expression XPath qui est évaluée pour donner une liste de valeurs. Chacune de ces valeurs est convertie en une chaîne de caractères. Lorsqu'une valeur est un nœud, la conversion retourne la valeur textuelle de celui-ci. Le texte est alors obtenu en concaténant ces différentes chaînes. Un espace `' '` est inséré, par défaut, entre ces chaînes. Le caractère inséré peut être changé en donnant une valeur à l'attribut `separator` de l'élément `xsl:value-of`.

La feuille se style suivante collecte des valeurs des enfants de l'élément racine `list` pour former le contenu d'un élément `values`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                          version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <values><xsl:value-of select="list/*" separator=","/></values>
  </xsl:template>
</xsl:stylesheet>
```

Le document suivant comporte un élément racine avec des enfants `item` contenant des valeurs `1`, `2` et `3`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<list><item>1</item><item>2</item><item>3</item></list>
```

En appliquant la feuille de style précédente au document précédent, on obtient le document suivant où apparaissent les trois valeurs `1`, `2` et `3` séparées par des virgules `','`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<values>1,2,3</values>
```

Le fonctionnement de `xsl:value-of` de XSLT 1.0 est différent et constitue une incompatibilité avec XSLT 2.0. Avec XSLT 1.0, seule la première valeur de la liste donnée par l'évaluation de l'expression de l'attribut`select` est prise en compte. Les autres valeurs sont ignorées. En appliquant la feuille de style précédente au document précédent avec un processeur XSLT 1.0, on obtient le document suivant où seule la valeur `1` apparaît.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<values>1</values>
```

Quelques exemples d'utilisation de `xsl:value-of`.

```
<xsl:value-of select="."/>
<xsl:value-of select="generate-id()"/>
<xsl:value-of select="key('idchapter', @idref)/title"/>
```

8.6.6. Élément et attribut

Tout élément contenu dans un élément `xsl:template` et n'appartenant pas à l'espace de nom `xsl` des feuilles de style est recopié à l'identique lors de l'application de la règle. Ceci permet d'ajouter facilement des éléments et des attributs dont les noms sont fixes. Il est parfois nécessaire d'ajouter des éléments et/ou des attributs dont les noms sont calculés dynamiquement. Les éléments `xsl:element` et `xsl:attribute` permettent respectivement de construire un nœud pour un élément ou un attribut. Le nom de l'élément ou de l'attribut est déterminé dynamiquement par l'attribut `name` de `xsl:element` et `xsl:attribute`. Cet attribut contient une expression XPath en attribut. L'évaluation des expressions XPath délimitées par des accolades `'{'` et `'}'` fournit le nom de l'élément ou de l'attribut. Le contenu de l'élément ou la valeur de l'attribut sont donnés par

l'attribut `select` ou par le contenu des éléments `xsl:element` et `xsl:attribute`. Ces deux possibilités s'excluent mutuellement. L'attribut `select` doit contenir une expression XPath.

Dans l'exemple suivant, le nom de l'élément est obtenu en concaténant la chaîne `'new-'` avec la valeur de la variable `var`.

```
<xsl:element name="{concat('new-', $var)}">...</element>
```

L'utilisation des éléments `xsl:element` et `xsl:element` est pratique lorsque l'apparition de l'élément ou de l'attribut est conditionnel. Le fragment de feuille style suivant construit un élément `tr` avec en plus un attribut`bgcolor` si la position de l'élément traité est paire.

```
<tr>
  <xsl:if test="position() mod 2 = 0">
    <xsl:attribute name="bgcolor">#ffffcc</xsl:attribute>
  </xsl:if>
  ...
</tr>
```

8.6.6.1. Groupe d'attributs

Il est fréquent qu'une feuille de style ajoute systématiquement les mêmes attributs à des éléments. Les *groupes d'attributs* définissent des ensembles d'attributs qu'il est, ensuite, facile d'utiliser pour ajouter des attributs aux éléments construits. Cette technique accroît, en outre, la flexibilité des feuilles de style. Le groupe peut être redéfini au niveau global, permettant ainsi, d'adapter la feuille de style sans modifier les règles.

Un groupe d'attributs est introduit par l'élément `xsl:attribute-set` dont l'attribut `name` précise le nom. Cet éléments doit être un enfant de l'élément racine `xsl:stylesheet` de la feuille de style. L'élément`xsl:attribute-set` contient des déclarations d'attributs introduites par des éléments `xsl:attribute`. Le groupe d'attribut est ensuite utilisé par l'intermédiaire de l'attribut `use-attribute-sets`. Cet attribut peut apparaître dans certains éléments XSLT comme `xsl:element` mais il peut également apparaître dans des éléments hors de l'espace de noms XSLT. Dans ce dernier cas, son nom doit être qualifié pour qu'il fasse partie de l'espace de noms XSLT. L'attribut `use-attribute-sets` contient une liste de noms de groupes d'attributs séparés par des espaces.

Un groupe d'attributs peut réutiliser d'autres groupes d'attributs. Il contient alors tous les attributs de ces groupes en plus de ceux qu'il déclare explicitement. Les

noms des groupes utilisés sont donnés par un attribut `use-attribute-sets` de l'élément `xsl:attribute-set`.

La feuille de style suivante permet la transformation d'un sous-ensemble, délibérément très réduit, de DocBook en XHTML. Le sous-ensemble est constitué des éléments `book`, `title`, `chapter`, `sect1` et `para`. La feuille de style définit trois groupes d'attributs de noms `text`, `title` et `para`. Le premier est prévu pour contenir des attributs généraux pour les éléments XHTML contenant du texte. Il est utilisé dans les définitions des deux autres groupes `title` et `para`. Le groupe `title` est utilisé pour ajouter des attributs aux éléments `h1` et `h2` construits par la feuille de style. Le groupe `para` est utilisé pour ajouter des attributs aux éléments `p`.

Il faut remarquer que de nom l'attribut `use-attribute-sets` n'est pas qualifié lorsque celui-ci apparaît dans un élément XSLT comme `xsl:attribute-set` alors qu'il est qualifié lorsque celui-ci apparaît dans des éléments hors de l'espace de noms XSLT comme `h1` et `p`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:dbk="http://docbook.org/ns/docbook"
                 xmlns="http://www.w3.org/1999/xhtml">
  <xsl:output ... />
  <!-- Définition d'un groupe d'attribut text -->
  <xsl:attribute-set name="text">
    <xsl:attribute name="align">left</xsl:attribute>
  </xsl:attribute-set>
  <!-- Définition d'un groupe d'attribut para pour les paragraphes -->
  <xsl:attribute-set name="para" use-attribute-sets="text"/>
  <!-- Définition d'un groupe d'attribut title pour les titres -->
  <xsl:attribute-set name="title" use-attribute-sets="text">
    <xsl:attribute name="id" select="generate-id()"/>
  </xsl:attribute-set>


  <xsl:template match="/">
    <xsl:comment>Generated by dbk2html.xsl</xsl:comment>
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title><xsl:value-of
select="dbk:book/dbk:title"/></title></head>
      <body><xsl:apply-templates/></body>
```

```
    </html>
  </xsl:template>
  <!-- Éléments non traités -->
  <xsl:template match="dbk:title|dbk:subtitle"/>
  <!-- Livre -->
  <xsl:template match="dbk:book">
    <xsl:apply-templates/>
  </xsl:template>
  <!-- Chapitres -->
  <xsl:template match="dbk:chapter">
    <h1                      xsl:use-attribute-sets="title"><xsl:value-of
select="dbk:title"/></h1>
    <xsl:apply-templates/>
  </xsl:template>
  <!-- Sections de niveau 1 -->
  <xsl:template match="dbk:sect1">
    <h2                      xsl:use-attribute-sets="title"><xsl:value-of
select="dbk:title"/></h2>
    <xsl:apply-templates/>
  </xsl:template>
  <!-- Paragraphes -->
  <xsl:template match="dbk:para">
    <p xsl:use-attribute-sets="para"><xsl:apply-templates/></p>
  </xsl:template>
</xsl:stylesheet>
```

Un document produit par cette feuille de style contient de multiples occurrences des mêmes attributs avec les mêmes valeurs. Il est préférable d'utiliser CSS qui permet de séparer le contenu de la présentation et de réduire, du même coup, la taille du fichier XHTML.

Une feuille de style peut importer la feuille de style précédente tout en redéfinissant certains groupes d'attributs pour en modifier le comportement.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                          version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Import de la feuille de style précédente -->
  <xsl:import href="dbk2html.xsl"/>
  <!-- Modification du groupe d'attributs text -->
```

```
  <xsl:attribute-set name="title">
    <!-- Nouvelle valeur de l'attribut align -->
    <xsl:attribute name="align">center</xsl:attribute>
  </xsl:attribute-set>
  <!-- Modification du groupe d'attributa text -->
  <xsl:attribute-set name="para">
    <!-- Ajout d'un d'attribut style -->
    <xsl:attribute name="style">font-family: serif</xsl:attribute>
  </xsl:attribute-set>
</xsl:stylesheet>
```

8.6.7. Commentaire et instruction de traitement

Les éléments `xsl:comment` et `xsl:processing-instruction` permettent respectivement de construire un nœud pour un commentaire ou une instruction de traitement. Le nom de l'instruction de traitement est déterminé dynamiquement par l'attribut `name` de `xsl:processing-instruction`. Cet attribut contient une expression XPath en attribut. Le texte du commentaire ou de l'instruction de traitement est donné par le contenu des éléments `xsl:comment` et `xsl:processing-instruction`.

```
<!-- Un commentaire dans la feuille de style -->
<xsl:comment>Un commentaire dans le document final</xsl:comment>
<xsl:processing-instruction name="dbhtml">
  filename="index.html"
</xsl:processing-instruction>
```

8.6.8. Liste d'objets

L'élément `xsl:sequence` construit une liste de valeurs déterminée par l'évaluation de l'expression XPath contenu dans l'attribut `select`.

Cet élément est souvent utilisé dans la déclaration d'une variable ou dans la définition d'une fonction d'extension.

8.6.9. Copie superficielle

L'élément `xsl:copy` permet de copier le nœud courant dans le document résultat. Cette copie est dite *superficielle* car elle copie seulement le nœud ainsi que les déclarations d'espaces de noms nécessaires. L'élément`xsl:copy` n'a pas d'attribut `select` car c'est toujours le nœud courant qui est copié.

Les attributs (dans le cas où le nœud courant est un élément) et le contenu ne sont pas copiés. Ils doivent être ajoutés explicitement. Le contenu de l'élément `xsl:copy` définit le contenu de l'élément copié. C'est ici qu'il faut, par exemple, ajouter des éléments `xsl:copy-of` ou `xsl:apply-templates`.

La feuille de style suivante transforme le document `bibliography.xml` en remplaçant chaque attribut `lang` d'un élément `book` par un enfant `lang` ayant comme contenu la valeur de l'attribut. La feuille de style est constituée de deux règles, une pour les éléments `book` et une autre pour tous les autres éléments. Dans les deux règles, la copie de l'élément est réalisée par un élément `xsl:copy`. La copie des attributs est réalisée par un élément `xsl:copy-of` qui sélectionne explicitement les attributs. La copie des enfants est réalisée par un élément `xsl:apply-templates` pour un appel récursif des règles.

```
<?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" indent="yes"/>
  <!-- Règle pour la racine et les éléments autres que book -->
  <xsl:template match="/|*">
    <xsl:copy>
      <!-- Copie explicite des attributs -->
      <xsl:copy-of select="@*"/>
      <!-- Copie explicite des enfants -->
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
  <!-- Règle pour les éléments book -->
  <xsl:template match="book">
    <xsl:copy>
      <!-- Copie explicite des attributs autres que 'lang' -->
      <xsl:copy-of select="@*[name() != 'lang']"/>
      <!-- Ajout de l'élément lang -->
      <lang><xsl:value-of select="@lang"/></lang>
      <!-- Copie explicite des enfants -->
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

En appliquant la feuille de style précédente au document `bibliography.xml`, on obtient le document suivant.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bibliography>
  <book key="Michard01">
    <lang>fr</lang>
    <title>XML langage et appplications</title>
    <author>Alain Michard</author>
    <year>2001</year>
    <publisher>Eyrolles</publisher>
    <isbn>2-212-09206-7</isbn>
    <url>http://www.editions-eyrolles/livres/michard/</url>
  </book>
  <book key="Zeldman03">
    <lang>en</lang>
    <title>Designing with web standards</title>
    <author>Jeffrey Zeldman</author>
    <year>2003</year>
    <publisher>New Riders</publisher>
    <isbn>0-7357-1201-8</isbn>
  </book>
  <!-- Fichier tronqué -->
  ...
</bibliography>
```

8.6.10. Copie profonde

L'élément `xsl:copy-of` permet de copier des nœuds sélectionnés ainsi que tous les descendants de ces nœuds dans le document résultat. Cette copie est dite *profonde* car tout le sous-arbre enraciné en un nœud sélectionné est copié. L'expression XPath contenue dans l'attribut `select` de `xsl:copy-of` détermine les nœuds à copier.

La feuille de style suivante transforme le document `bibliography.xml` en répartissant la bibliographie en deux parties dans des éléments `bibliodiv` contenant respectivement les livres en français et en anglais. L'unique règle de la feuille de style crée le squelette avec les deux éléments `bibliodiv` et copie les livres dans ces deux éléments grâce à deux éléments `xsl:copy-of`.

```
<?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" indent="yes"/>

  <xsl:template match="/">

    <bibliography>

      <bibliodiv>

        <xsl:copy-of select="bibliography/book[@lang='fr']"/>

      </bibliodiv>

      <bibliodiv>

        <xsl:copy-of select="bibliography/book[@lang='en']"/>

      </bibliodiv>

    </bibliography>

  </xsl:template>

</xsl:stylesheet>
```

8.6.11. Numérotation

Le rôle de l'élément `xsl:number` est double. Sa première fonction est de créer un entier ou une liste d'entiers pour numéroter un élément. La seconde fonction est de formater cet entier ou cette liste. La seconde fonction est plutôt adaptée à la numérotation. Pour formater un nombre de façon précise, il est préférable d'utiliser la fonction XPath `format-number()`.

8.6.11.1. Formats

La fonction de formatage est relativement simple. L'attribut `format` de `xsl:number` contient une chaîne formée d'un *préfixe*, d'un indicateur de format et d'un *suffixe*. Le préfixe et le suffixe doivent être formés de caractères non alphanumériques. Ils sont recopiés sans changement. L'indicateur de format est remplacé par l'entier. Le tableau suivant récapitule les différents formats possibles. Le format par défaut est `1`.

Format	Résultat
1	1, 2, 3, …, 9, 10, 11, …
01	01, 02, 03, …, 09, 10, 11, …
a	a, b, c, …, z, aa, ab, …
A	A, B, C, …, Z, AA, AB, …
i	i, ii, iii, iv, v, vi, vii, viii, ix, x, xi, …
I	I, II, III, IV, V, VI, VII, VIII, IX, X, XI, …

Tableau 8.1. Formats de `xsl:number`

Il existe aussi des formats `w`, `W` et `Ww` permettant d'écrire les nombres en toutes lettres. L'attribut `lang` qui prend les mêmes valeurs que l'attribut `xml:lang` spécifie la langue dans laquelle sont écrits les nombres. Il semblerait que l'attribut `lang` ne soit pas pris en compte.

8.6.11.2. Calcul du numéro

Le numéro calculé par `xsl:number` peut être donné de façon explicite par l'attribut `value` qui contient une expression XPath. L'évaluation de cette expression fournit le nombre résultat. Cette méthode permet d'avoir un contrôle total sur le numéro. Si l'attribut `value` est absent, le numéro est calculé grâce aux valeurs des attributs `level`, `count` et `from`. L'attribut `level` détermine le mode de calcul alors que les attributs `count` et `from` les éléments pris en compte. Chacun de ces trois attributs contient un motif XPath permettant de sélectionner des nœuds.

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

L'attribut `level` peut prendre les valeurs `single`, `multiple` et `any`. Les modes de calcul `single` et `any` fournissent un seul entier alors que le mode `multiple` fournit une liste d'entiers. Dans ce dernier cas, le format peut contenir plusieurs indicateurs de formats séparés par des caractères non alphanumériques comme `1.1.1` ou `[A-1-i]`.

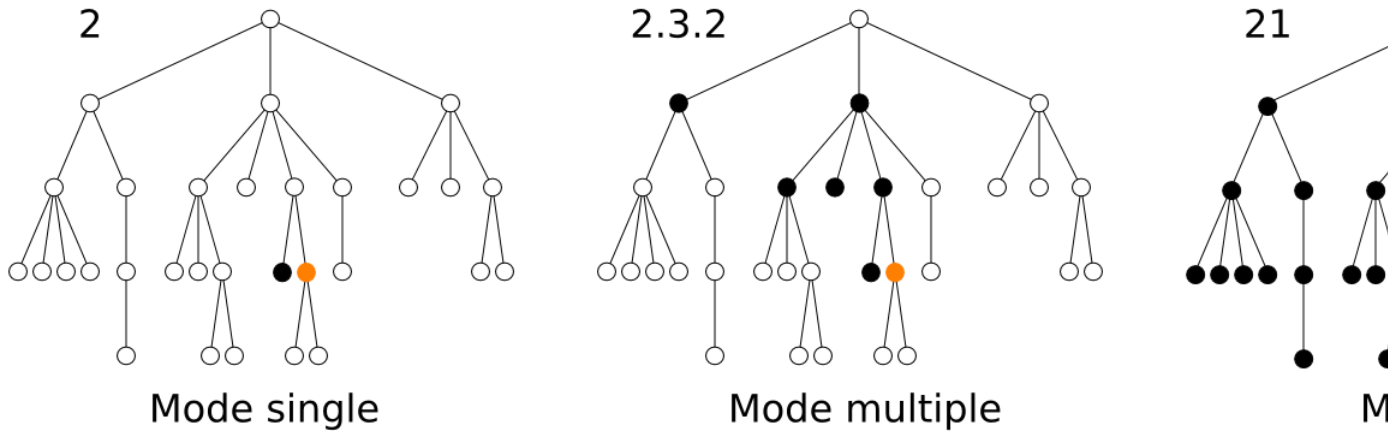

Figure 8.3. Modes de `xsl:number`

Dans le mode `single`, le numéro est égal au nombre (augmenté d'une unité pour commencer avec 1) de frères gauches du nœud courant qui satisfont le motif donné par `count`. Rappelons qu'un *frère* est un enfant du même nœud père et qu'il est *gauche* s'il précède le nœud courant dans le document.

La feuille de style suivante ajoute un numéro aux éléments `section`. Ce numéro est calculé dans le mode `single`.

```
<?xml version="1.0" encoding="us-ascii"?>
<xsl:stylesheet                                          version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="*">
    <xsl:copy>
      <xsl:if test="name()='section'">
        <!-- format="1" est la valeur par défaut -->
        <xsl:number level="single" count="section" format="1"/>
      </xsl:if>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

On considère le document XML suivant qui représente le squelette d'un livre avec des chapitres, des sections et des sous-sections.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<book>
  <chapter>
    <section>
      <section></section><section></section>
    </section>
    <section>
      <section></section><section></section>
    </section>
  </chapter>
  <chapter>
    <section>
      <section></section><section></section>
    </section>
    <section>
      <section></section><section></section>
    </section>
  </chapter>
```

```
</book>
```

En appliquant la feuille de style au document précédent, on obtient le document XML suivant. Chaque élément `section` contient en plus un numéro calculé par `xsl:number` en mode `single`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<book>
  <chapter>
    <section>1
      <section>1</section><section>2</section>
    </section>
    <section>2
      <section>1</section><section>2</section>
    </section>
  </chapter>
  <chapter>
    <section>1
      <section>1</section><section>2</section>
    </section>
    <section>2
      <section>1</section><section>2</section>
    </section>
  </chapter>
</book>
```

Dans le mode `multiple`, l'élément `xsl:number` fournit une liste d'entiers qui est calculée de la façon suivante. Le *nœud de départ* est déterminé par l'attribut `from` qui contient un motif XPath. C'est l'ancêtre le plus proche du nœud courant qui satisfait le motif de l'attribut `from`. Ensuite, on considère chacun des ancêtres entre le nœud de départ et le nœud courant qui satisfait l'attribut `count`. Pour chacun de ces ancêtres, le nombre (plus une unité) de frères gauches qui satisfont le motif de `count` fournit un des entiers de la suite.

Si l'élément `xsl:number` de la feuille de style précédente est remplacé par l'élément suivant, on obtient le document ci-dessous. Comme le format est `A.1.i`, chaque section contient un numéro global formé d'un numéro de chapitre (`A`, `B`, …), d'un numéro de section (`1`, `2`, …) et d'un numéro de sous-section (`i`, `ii`, …). Ces différents numéros sont séparés par les points `'.'` qui sont repris du format.

```
 <xsl:number level="multiple" count="chapter|section" format="A.1.i"/>
<?xml version="1.0" encoding="iso-8859-1"?>
<book>
  <chapter>
    <section>A.1
      <section>A.1.i</section><section>A.1.ii</section>
    </section>
    <section>A.2
      <section>A.2.i</section><section>A.2.ii</section>
    </section>
  </chapter>
  <chapter>
    <section>B.1
      <section>B.1.i</section><section>B.1.ii</section>
    </section>
    <section>B.2
      <section>B.2.i</section><section>B.2.ii</section>
    </section>
  </chapter>
</book>
```

Dans le mode `any`, le *nœud de départ* est égal au dernier nœud avant le nœud courant qui vérifie le motif donné par l'attribut `from`. Par défaut le nœud de départ est la racine du document. Le numéro est égal au nombre (augmenté d'une unité pour commencer avec 1) de nœuds entre le nœud de départ et le nœud courant qui satisfont le motif donné par l'attribut `count`.

Si l'élément `xsl:number` de la feuille de style précédente est remplacé par l'élément suivant, on obtient le document ci-dessous. Chaque section contient son numéro d'ordre dans le document car la valeur par défaut de `from` est la racine du document.

```
<xsl:number level="any" count="section" format="1"/>
<?xml version="1.0" encoding="iso-8859-1"?>
<book>
  <chapter>
    <section>1
      <section>2</section><section>3</section>
    </section>
    <section>4
```

```
      <section>5</section><section>6</section>
    </section>
  </chapter>
  <chapter>
    <section>7
      <section>8</section><section>9</section>
    </section>
    <section>10
      <section>11</section><section>12</section>
    </section>
  </chapter>
</book>
```

L'élément `xsl:number` suivant utilise l'attribut `from` pour limiter la numérotation des éléments `section` aux contenus des éléments `chapter`. En appliquant la feuille de style précédente avec cet élément `xsl:number`, on obtient le document ci-dessous. Chaque section contient son numéro d'ordre dans le chapitre.

```
<xsl:number level="any" count="section" from="chapter" format="1"/>
<?xml version="1.0" encoding="iso-8859-1"?>
<book>
  <chapter>
    <section>1
      <section>2</section><section>3</section>
    </section>
    <section>4
      <section>5</section><section>6</section>
    </section>
  </chapter>
  <chapter>
    <section>1
      <section>2</section><section>3</section>
    </section>
    <section>4
      <section>5</section><section>6</section>
    </section>
  </chapter>
</book>
```

8.6.12. Formatage de nombres

L'élément `xsl:number` a des possibilités limitées pour formater un nombre de manière générale. Ils possède deux attributs `grouping-separator` et `grouping-size` dont les valeurs par défaut sont respectivement `','` et `3`. Dans l'exemple suivant, l'entier 1234567 est formaté en `1.234.567`.

```
<xsl:number grouping-separator="." value="12345678"/>
```

La fonction XPath `format-number()` permet de formater un nombre entier ou décimal de façon plus précise. Le premier paramètre est le nombre à formater et le second est une chaîne de caractères qui décrit le formatage à effectuer. Cette fonction est inspirée de la classe `DecimalFormat` de Java et elle s'apparente, par les formats utilisés, à la fonction `printf` du langage C. Un troisième paramètre optionnel référence éventuellement un élément `xsl:decimal-format` pour changer la signification de certains caractères dans le format.

Lorsqu'aucun élément `xsl:decimal-format` n'est référencé, le format est une chaîne de caractères formée des caractères `'#'`, `'0'`, `'.'`, `','`, `'%'`, `'‰'` (de code hexadécimal `x2030`) et `';'`. La signification de ces différents caractères est donnée ci-dessous. La chaîne passée en second paramètre peut aussi contenir d'autres caractères qui sont recopiés inchangés dans le résultat.

`'#'`

position pour un chiffre

`'0'`

position pour un chiffre remplacé éventuellement par `0`

`'.'`

position du point décimal

`','`

position du séparateur de groupe (milliers, millions, …)

`';'`

séparateur entre un format pour les nombres positifs et un format pour les nombres négatifs.

La chaîne de caractères passée en second paramètre à `format-number()` peut contenir deux formats séparés par un caractère `';'` comme `#000;-#00` par exemple. Le premier format `#000` est alors utilisé pour les nombres positifs et le second

format `-#00` pour les nombres négatifs. Dans ce cas, le second format doit explicitement insérer le caractère `'-'` car c'est la valeur absolue du nombre qui est formatée. En formatant les nombres 12 et -12 avec ce format `#000;-#00`, on obtient respectivement `012` et `-12`. La table ci-dessous donne quelques exemples de résultats de la fonction `format-number()` avec des formats différents.

Nombre\Format	##	####	#,#00.##	####.00	0000.00
1	1	1	01	1.00	0001.00
123	123	123	123	123.00	0123.00
1234	1234	1234	1,234	1234.00	1234.00
12.34	12	12	12.34	12.34	0012.34
1.234	1	1	01.23	1.23	0001.23

Tableau 8.2. Résultats de `format-number()`

Les caractères `'%'` et `'‰'` permettent de formater une fraction entre 0 et 1 comme un pourcentage ou un millième. Le formatage du nombre 0.1234 avec les formats `#%`, `#.##%` et `#‰` donne respectivement `12%`, `12.34%` et`123‰`.

L'élément `xsl:decimal-format` permet de changer les caractères utilisés dans le format. Cet élément déclare un objet qui définit l'interprétation des caractères dans le format utilisé par `format-number()`. L'attribut`name` donne le nom de l'objet qui est utilisé comme troisième paramètre de la fonction `format-number()`. Outre cet attribut, l'élément `xsl:decimal-format` possède plusieurs attributs permettant de spécifier les caractères utilisés pour marquer les différentes parties du nombre (point décimal, séparateur de groupe, etc …).

`decimal-separator`

caractère pour marquer le point décimal (`'.'` par défaut)

`grouping-separator`

caractère pour séparer les groupes (`','` par défaut)

`digit`

caractère pour la position d'un chiffre (`'#'` par défaut)

`zero-digit`

caractère pour la position d'un chiffre remplacé par `'0'` (`'0'` par défaut)

`pattern-separator`

caractère pour séparer deux formats pour les nombres positifs et les nombres négatifs (`';'` par défaut)

`percent`

caractère pour formater les pourcentages (`'%'` par défaut)

`per-mille`

caractère pour formater les millièmes (`'‰'` par défaut)

Les éléments `xsl:decimal-format` doivent être enfants de l'élément racine `xsl:stylesheet` de la feuille de style et leur portée est globale. Ils peuvent être référencés par la fonction `format-number()` dans n'importe quelle expression XPath de la feuille de style.

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

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

  <!-- Format des nombres en englais -->

  <xsl:decimal-format name="en" decimal-separator="."

                                grouping-separator=","/>

  <!-- Format des nombres en français -->

  <xsl:decimal-format name="fr" decimal-separator=","

                                grouping-separator="."/>

  ...

  <price xml:lang="en-GB" currency="pound">

    <xsl:value-of     select="format-number($price,     '###,###,###.##',
'en')"/>

  </price>

  ...

  <price xml:lang="fr" currency="euro">

    <xsl:value-of     select="format-number($price,     '###.###.###,##',
'fr')"/>

  </price>

  ...
```

8.7. Structures de contrôle

Le langage XSLT propose, comme tout langage de programmation, des structures de contrôle permettant d'effectuer des tests et des boucles. Il contient les deux éléments `xsl:if` et `xsl:choose` pour les tests et les deux éléments `xsl:for-each` et `xsl:for-each-group` pour les boucles. L'élément `xsl:if` autorise un test sans

alternative (pas d'élément `xsl:else`) alors que l'élément `xsl:choice` permet, au contraire un choix entre plusieurs alternatives. L'élément `xsl:for-each` pemet des boucles simples sur des nœuds sélectionnés. L'élément `xsl:for-each-group` permet de former des groupes à partir de nœuds sélectionnés puis de traiter successivement les différents groupes. Certaines constructions XSLT sont parfois réalisées de manière plus concise par des structures de contrôle XPath placées dans des attributs.

8.7.1. Conditionnelle sans alternative

L'élément `xsl:if` permet de réaliser un test. De façon surprenante, cet élément ne propose pas d'alternative car il n'existe pas d'élément `xsl:else`. Lorsqu'une alternative est nécessaire, il faut utiliser l'élément`xsl:choose`.

La condition du test est une expression XPath contenue dans l'attribut `test` de `xsl:if`. Cette expression est évaluée puis convertie en valeur booléenne. Si le résultat est `true`, le contenu de l'élément `xsl:if` est pris en compte. Sinon, le contenu de l'élément `xsl:if` est ignoré.

```
<xsl:if test="not(position()=last())">
  <xsl:text>, </xsl:text>
</xsl:if>
```

8.7.2. Conditionnelle à alternatives multiples

L'élément `xsl:choose` permet de réaliser plusieurs tests consécutivement. Il contient des éléments `xsl:when` et éventuellement un élément `xsl:otherwise`. Chacun des éléments `xsl:when` possède un attribut `test`contenant une expression XPath servant de condition.

Les conditions contenues dans les attributs `test` des éléments `xsl:when` sont évaluées puis converties en valeur booléenne dans l'ordre des éléments `xsl:when` Le contenu du permier élément `xsl:when` dont la condition donne la valeur `true` est pris en compte et les contenus des autres `xsl:when` et d'un éventuel `xsl:otherwise` sont ignorés. Si aucune condition ne donne la valeur `true`, le contenu de l'élément `xsl:otherwise` est pris en compte et les contenus de tous les éléments `xsl:when` sont ignorés.

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

```
<xsl:choose>
  <xsl:when test="title">
    <xsl:value-of select="title"/>
```

```
  </xsl:when>
  <xsl:otherwise>
    <xsl:text>Section </xsl:text>
    <xsl:number level="single" count="section"/>
  </xsl:otherwise>
</xsl:choose>
```

8.7.3. Itération simple

L'élément `xsl:for-each` permet de réaliser des boucles en XSLT. L'itération est déjà présente implicitement avec l'élément `xsl:apply-templates` puisqu'une règle est successivement appliquée à chacun des nœuds sélectionnés. L'élément `xsl:for-each` réalise une boucle de manière explicite.

L'attribut `select` détermine les objets traités. L'expression XPath qu'il contient est évaluée pour donner une liste `l` d'objets qui est, ensuite, parcourue. Le contenu de l'élément `xsl:for-each` est exécuté pour chacun des objets de la liste `l`. Le focus est modifié pendant l'exécution de `xsl:for-each`. À chaque itération, l'objet courant est fixé à un des objets de la liste `l`. La taille du contexte est également fixée à la longueur de `l` et la position dans le contexte est finalement fixée à la position de l'objet courant dans la liste `l`.

La feuille de style suivante présente la bibliographie `bibliography.xml` sous forme d'un tableau XHTML. Le tableau est construit par l'unique règle de la feuille de style. La première ligne du tableau avec des éléments `th`est ajoutée explicitement et les autres lignes avec des éléments `td` sont ajoutées par un élément `xsl:for-each` qui parcourt tous les éléments `book` de `bibliography.xml`. Le résultat de la fonction `position()` est formatée par l'élément `xsl:number` pour numéroter les lignes du tableau.

```
<?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>Bibliographie en tableau</title>
      </head>
      <body>
        <h1>Bibliographie en tableau</h1>
        <table align="center" border="1" cellpadding="2" cellspacing="0">
          <tr>
```

```
            <th>Numéro</th>
            <th>Titre</th>
            <th>Auteur</th>
            <th>Éditeur</th>
            <th>Année</th>
          </tr>
          <xsl:for-each select="bibliography/book">
            <xsl:sort select="author" order="ascending"/>
            <tr>
              <td><xsl:number value="position()" format="1"/></td>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="author"/></td>
              <td><xsl:value-of select="publisher"/></td>
              <td><xsl:value-of select="year"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

En appliquant la feuille de style précédente au document `bibliography.xml`, on obtient le document suivant.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Bibliographie en tableau</title>
  </head>
  <body>
    <h1>Bibliographie en tableau</h1>
    <table align="center" border="1" cellpadding="2" cellspacing="0">
      <tr>
        <th>Num&eacute;ro</th>
        <th>Titre</th>
        <th>Auteur</th>
        <th>&Eacute;diteur</th>
```

```
        <th>Ann&eacute;e</th>
      </tr>
      <tr>
        <td>1</td>
        <td>XML langage et appplications</td>
        <td>Alain Michard</td>
        <td>Eyrolles</td>
        <td>2001</td>
      </tr>
      <!-- Fichier tronqué -->
      ...
    </table>
  </body>
</html>
```

8.7.4. Itération sur des groupes

L'élément `xsl:for-each-group` est un ajout de XSLT 2.0. Il permet de grouper des nœuds du document source suivant différents critères puis de parcourir les groupes formés.

La feuille de style suivante donne un premier exemple simple d'utilisation de l'élément `xsl:for-each-group`. Elle présente la bibliographie en XHTML en regroupant les livres par années. Le regroupement est réalisé par l'élément `xsl:for-each-group` avec l'attribut `group-by` égal à l'expression XPath `year`. L'attribut `select` détermine que les éléments à regrouper sont les élément `book`. Le traitement de chacun des groupes est réalisé par le contenu de l'élément `xsl:for-each-group`. La clé du groupe est d'abord récupérée par la fonction `current-grouping-key()` pour construire le titre contenu dans l'élément XHTML `h2`. Les éléments `book` de chaque groupe sont ensuite traités, l'un après l'autre, grâce à un élément `xsl:for-each`. L'attribut `select` de cet élément utilise la fonction `current-group()` qui retourne la liste des objets du groupe.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<xsl:output method="xhtml" encoding="iso-8859-1" indent="yes"/>
<xsl:template match="/">
<html>
```

```
  <head>
    <title>Bibliographie par année</title>
  </head>
  <body>
  <h1>Bibliographie par année</h1>
  <!-- Regroupement des livres par années -->
  <xsl:for-each-group select="bibliography/book" group-by="year">
    <!-- Tri des groupes par années -->
    <xsl:sort select="current-grouping-key()"/>
    <!-- Titre avec l'année -->
    <h2>
      <xsl:text>Année </xsl:text>
      <xsl:value-of select="current-grouping-key()"/>
    </h2>
    <!-- Liste des livres de l'année -->
    <ul>
      <!-- Traitement de chacun des éléments du groupe -->
      <xsl:for-each select="current-group()">
        <!-- Tri des éléments du groupe par auteur puis publisher -->
        <xsl:sort select="author"/>
        <xsl:sort select="publisher"/>
        <xsl:apply-templates/>
      </xsl:for-each>
    </ul>
  </xsl:for-each-group>
  </body>
</html>
</xsl:template>
<!-- Règle pour les éléments title -->
<xsl:template match="title">
  <i><xsl:apply-templates/></i>
  <xsl:call-template name="separator"/>
</xsl:template>
<!-- Règle pour les autres éléments -->
<xsl:template match="*">
  <xsl:apply-templates/>
  <xsl:call-template name="separator"/>
```

```
</xsl:template>
<!-- Virgule après les éléments -->
<xsl:template name="separator">
  <xsl:if test="position() != last()">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

En appliquant la feuille de style précédente au document `bibliography.xml`, on obtient le document suivant.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1" />
    <title>Bibliographie par année</title>
  </head>
  <body>
    <h1>Bibliographie par année</h1>
    <h2>Année 2000</h2>
    <ul>
      <li><i>XML by Example</i>, Benoît Marchal, 2000,
        Macmillan Computer Publishing, 0-7897-2242-9</li>
      ...
    </ul>

    <h2>Année 2001</h2>
    ...
  </body>
</html>
```

La feuille de style suivante donne un exemple classique d'utilisation de l'élément `xsl:for-each-group`. Celle-ci effectue une transformation d'un document XHTML en un document DocBook. Pour simplifier, on se contente de sous-ensembles très restreints de ces deux dialectes XML. On considère uniquement les éléments `html`, `body`, `h1`, `h2` et `p` de XHTML et des éléments `book`, `chapter`, `sect1`, `title` et `para` de DocBook. Ces deux langages

organisent un document de manières différentes. Dans un document XHTML, les chapitres et les sections sont uniquement délimités par les titres `h1` et `h2`. Au contraire, dans un document DocBook, les éléments `chapter` et `sect1` encapsulent les chapitres et les sections. Ces différences rendent plus difficile la transformation de XHTML vers DocBook. Pour trouver le contenu d'un chapitre, il est nécessaire de regrouper tous les éléments placés entre deux éléments `h1` consécutifs. L'élément `xsl:for-each-group` permet justement de réaliser facilement cette opération.

L'espace de noms par défaut de la feuille de style est celui de DocBook. Les noms des éléments XHTML doivent ainsi être qualifiés par le préfixe `html` associé à l'espace de noms de XHTML.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:html="http://www.w3.org/1999/xhtml"
                xmlns="http://docbook.org/ns/docbook"
                exclude-result-prefixes="xsl html">
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="/">
    <book>
      <xsl:apply-templates select="html:html/html:body"/>
    </book>
  </xsl:template>
  <xsl:template match="html:body">
    <!-- Regroupement des éléments avec l'élément h1 qui précède -->
    <xsl:for-each-group select="*" group-starting-with="html:h1">
      <chapter>
        <!-- Titre avec le contenu de l'élément h1 -->
        <title><xsl:value-of select="current-group()[1]"/></title>
        <!-- Traitement du groupe -->
        <!-- Regroupement des éléments avec l'élément h2 qui précède -->
        <xsl:for-each-group select="current-group()"
                            group-starting-with="html:h2">
          <xsl:choose>
            <xsl:when test="local-name(current-group()[1]) = 'h2'">
              <sect1>
                <!-- Titre avec le contenu de l'élément h2 -->
```

```
                <title><xsl:value-of                    select="current-
group()[1]"/></title>
                <xsl:apply-templates select="current-group()"/>
              </sect1>
            </xsl:when>
            <xsl:otherwise>
              <xsl:apply-templates select="current-group()"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:for-each-group>
      </chapter>
    </xsl:for-each-group>
  </xsl:template>
  <!-- Supression des éléments h1 et h2 -->
  <xsl:template match="html:h1|html:h2"/>
  <!-- Transformation des éléments p en éléments para -->
  <xsl:template match="html:p">
    <para><xsl:apply-templates/></para>
  </xsl:template>
</xsl:stylesheet>
```

Le document suivant est le document XHTML sur lequel est appliqué la tranformation. Celui-ci représente le squelette typique d'un document XHTML avec des titres de niveaux 1 et 2 et des paragraphes.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <title>Fichier HTML exemple</title>
  </head>
  <body>
  <h1>Titre I</h1>
      <p>Paragraphe I.0.1</p>
    <h2>Titre I.1</h2>
      <p>Paragraphe I.1.1</p>
      <p>Paragraphe I.1.2</p>
    <h2>Titre I.2</h2>
      <p>Paragraphe I.2.1</p>
```

```
      <p>Paragraphe I.2.2</p>
  <h1>titre II</h1>
      <p>Paragraphe II.0.1</p>
    <h2>Titre II.1</h2>
      <p>Paragraphe II.1.1</p>
      <p>Paragraphe II.1.2</p>
    <h2>Titre II.2</h2>
      <p>Paragraphe II.2.1</p>
      <p>Paragraphe II.2.2</p>
  </body>
</html>
```

Le document suivant est le document DocBook obtenu par transformation par la feuille de style prédédente du document XHTML précédent. Les deux éléments `h1` du document XHTML donnent deux éléments `chapter`dans le document DocBook.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<book xmlns="http://docbook.org/ns/docbook">
  <chapter>
    <title>Titre I</title>
    <para>Para I.0.1</para>
    <sect1>
      <title>Titre I.1</title>
      <para>Para I.1.1</para>
      <para>Para I.1.2</para>
    </sect1>
    <sect1>
      <title>Titre I.2</title>
      <para>Para I.2.1</para>
      <para>Para I.2.2</para>
    </sect1>
  </chapter>
  <chapter>
    <title>titre II</title>
    <para>Para II.0.1</para>
    <sect1>
      <title>Titre II.1</title>
```

```
      <para>Para II.1.1</para>
      <para>Para II.1.2</para>
    </sect1>
    <sect1>
      <title>Titre II.2</title>
      <para>Para II.2.1</para>
      <para>Para II.2.2</para>
   </sect1>
  </chapter>
</book>
```

8.8. Tris

L'élément `xsl:sort` permet de trier des éléments avant de les traiter. L'élément `xsl:sort` doit être le premier fils des éléments `xsl:apply-templates`, `xsl:call-template`, `xsl:for-each` ou `xsl:for-each-group`. Le tri s'applique à tous les éléments sélectionnés par l'attribut `select` de ces différents éléments.

Le fragment de feuille de style suivant permet par exemple de trier les éléments `book` par auteur par ordre croissant.

```
<xsl:apply-templates select="bibliography/book">
  <xsl:sort select="author" order="ascending"/>
</xsl:apply-templates>
```

L'attribut `select` de `xsl:sort` détermine la clé du tri. L'attribut `data-type` qui peut prendre les valeurs `number` ou `text` spécifie comment les clés doivent être interprétées. Il est possible d'avoir plusieurs clés de tri en mettant plusieurs éléments `xsl:sort` comme dans l'exemple suivant. Les éléments `book` sont d'abord triés par auteur puis par année.

```
<xsl:apply-templates select="bibliography/book">
  <xsl:sort select="author" order="ascending"/>
  <xsl:sort select="year"   order="descending"/>
</xsl:apply-templates>
```

Le tri réalisé par `xsl:sort` est basé sur les valeurs retournées par l'expression XPath contenue dans l'attribut `select`. Cette expression est souvent le nom d'un enfant ou d'un attribut mais elle peut aussi être plus complexe. La feuille de style suivante réordonne les enfants des éléments `book`. L'expression contenue dans

l'attribut `select` de `xsl:sort` retourne un numéro d'ordre en fonction du nom de l'élément. Ce numéro est calculé avec la fonction `index-of()` et une liste de noms dans l'ordre souhaité. Cette solution donne une expression concise. Elle a aussi l'avantage que l'ordre est donné par une liste qui peut être fixe ou calculée. Cette liste peut, par exemple, être la liste des noms des enfants du premier élément `book`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes"/>

  <!-- Liste des noms des enfants du premier élément book -->
  <xsl:variable name="orderlist" as="xsd:string*"
                select="/bibliography/book[1]/*/name()"/>
  ...
  <xsl:template match="book">
    <xsl:copy>
      <!-- Copie des attributs -->
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates>
       <!-- Tri des enfants dans l'ordre donné par la liste fixe -->
       <!-- Les noms absents de la liste sont placés à la fin -->
       <xsl:sort select="(index-of(('title', 'author', 'publisher',
                                    'year', 'isbn'), name()),10)[1]"/>
       <!-- Tri dans l'ordre des enfants du premier élément book -->
       <!-- <xsl:sort select="index-of($orderlist, name())"/> -->
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

8.8.1. Tri de listes

L'élément `xsl:perform-sort` permet d'appliquer un tri à une suite quelconque d'objets, en particulier avant de l'affecter à une variable. Ses enfants doivent être un ou des éléments `xsl:sort` puis des éléments qui construisent la suite.

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

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsl:output method="text" encoding="iso-8859-1"/>
  <xsl:template match="/">
    <xsl:variable name="list" as="xsd:integer*">
      <xsl:perform-sort>
        <xsl:sort data-type="number" order="ascending"/>
        <xsl:sequence select="(3, 1, 5, 0)"/>
      </xsl:perform-sort>
    </xsl:variable>
    <!-- Produit 0,1,3,5 -->
    <xsl:value-of select="$list" separator=","/>
  </xsl:template>
</xsl:stylesheet>
```

8.9. Variables et paramètres

Le langage XSLT permet l'utilisation de variables pouvant stocker des valeurs. Les valeurs possibles comprennent une valeur atomique, un nœud ou une suite de ces valeurs, c'est-à-dire toutes les valeurs des expressions XPath. Les variables peuvent être utilisées dans les expressions XPath.

Le langage XSLT distingue les variables des paramètres. Les variables servent à stoker des valeurs intermédiaires alors que les paramètres servent à transmettre des valeurs aux règles. Les variables sont introduites par l'élément `xsl:variable` et les paramètres par l'élément `xsl:param`. L'élément `xsl:with-param` permet d'instancier un paramètre lors de l'appel à une règle.

8.9.1. Variables

La valeur de la variable est fixée au moment de sa déclaration par l'élément `xsl:variable` et ne peut plus changer ensuite. Les variables ne sont donc pas vraiment *variables*. Il s'agit d'objets *non mutables* dans la terminologie des langages de programmation. La portée de la variable est l'élément XSLT qui la contient. Les variables dont la déclaration est enfant de l'élément `xsl:stylesheet` sont donc globales.

L'attribut `name` détermine le nom de la variable. La valeur est donnée soit par une expression XPath dans l'attribut `select` soit directement dans le contenu de l'élément `xsl:variable`. Un attribut optionnel `as` peut spécifier le type de la variable. Les types possibles sont les types XPath. Dans l'exemple suivant, les deux

variables `squares` et `cubes` sont déclarées de type `xsd:integer*`. Chacune d'elles contient donc une liste éventuellement vide d'entiers. La valeur de la variable `square` est donnée par l'élément `xsl:sequence` contenu dans l'élément `xsl:variable`. La valeur de la variable `cubes` est donnée par l'expression XPath de l'attribut`select`.

```
<xsl:variable name="squares" as="xsd:integer*">
  <xsl:for-each select="1 to 5">
    <xsl:sequence select=". * ."/>
  </xsl:for-each>
</xsl:variable>
<xsl:variable name="cubes" as="xsd:integer*"
              select="for $i in 1 to 5 return $i * $i * $i"/>
```

Une variable déclarée peut apparaître dans une expression XPath en étant précédée du caractère `'$'` comme dans l'exemple suivant.

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

Une variable permet aussi de mémoriser un ou plusieurs nœuds. Il est parfois nécessaire de mémoriser le nœud courant dans une variable afin de pouvoir y accéder dans une expression XPath qui modifie contexte dynamique. Le fragment de feuille de style mémorise le nœud courant dans la variable `current`. Elle l'utilise ensuite pour sélectionner les éléments `publisher` dont l'attribut `id` est égal à l'attribut `by` du nœud courant.

```
<xsl:variable name="current" select="."/>
<xsl:xsl:copy-of select="//publisher[@id = $current/@by]"/>
```

XSLT 2.0 a introduit une fonction `current()` qui retourne le nœud courant. Il n'est plus nécessaire de le stocker dans une variable. L'exemple précédent pourrait être réécrit de la façon suivante.

```
<xsl:xsl:copy-of select="//publisher[@id = current()/@by]"/>
```

L'expression XPath `//publisher[@id = $current/@by]` n'est pas très efficace car elle nécessite un parcours complet du document pour retrouver le bon élément `publisher`. Elle peut avantageusement être remplacée par un appel à la fonction `key()`. Il faut au préalable créer avec `xsl:key` un index des éléments `publisher` par leur attribut `id`. Cette approche est développée dans l'exemple suivant.

Une variable peut aussi être utilisée, dans un souci d'efficacité pour mémoriser un résultat intermédiaire. Dans l'exemple suivant, le nœud retourné par la fonction `key()` est mémorisé dans la variable `result` puis utilisé à plusieurs reprises.

```
<!-- Indexation des éléments publisher par leur attribut id -->
<xsl:key name="idpublisher" match="publisher" use="@id"/>
...
<!-- Sauvegarde du noeud recherché -->
<xsl:variable name="result" select="key('idpublisher', @by)"/>
<publisher>
  <!-- Utilisation multiple du noeud -->
  <xsl:copy-of select="$result/@*[name() != 'id']"/>
  <xsl:copy-of select="$result/* | $result/text()"/>
</publisher>
```

L'élément `xsl:variable` permet également de déclarer des variables locales lors de la définition de fonctions d'extension XPath.

8.9.2. Paramètres

Il existe des paramètres XSLT qui s'apparentent aux paramètres des fonctions des langages classiques comme C ou Java. Ils servent à transmettent des valeurs à la feuille de style et aux règles. Les paramètres sont déclarés par l'élément `xsl:param` qui permet également de donner une valeur par défaut comme en C++. Cet élément peut être enfant de l'élément racine `xsl:stylesheet` ou des éléments `xsl:template`. Dans le premier cas, le paramètre est *global* et dans le second cas, il est *local* à la règle déclarée par `xsl:template`.

Comme avec l'élément `xsl:variable`, l'attribut `name` de l'élément `xsl:param` détermine le nom du paramètre. La valeur par défaut est optionnelle. Elle est donnée soit par une expression XPath dans l'attribut `select` soit directement dans le contenu de l'élément `xsl:param`. Un attribut optionnel `as` peut spécifier le type du paramètre. Le fragment suivant déclare un paramètre `bg-color` avec une valeur par défaut égale à la chaîne de caractères `white`.

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

Les apostrophes `'''` sont nécessaires autour de la chaîne `white` car la valeur de l'attribut `select` est une expression XPath. La même déclaration peut également prendre la forme suivante sans les apostrophes.

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

8.9.2.1. Paramètres globaux

Les paramètres globaux sont déclarés par un élément `xsl:param` enfant de l'élément `xsl:stylesheet`. Leur valeur est fixée au moment de l'appel au processeur XSLT. Leur valeur reste constante pendant toute la durée du traitemet et ils peuvent être utilisés dans toute la feuille de style. La syntaxe pour fixer la valeur d'un paramètre global dépend du processeur XSLT. Les processeurs qui peuvent être utilisés en ligne de commande ont généralement une option pour donner une valeur à un paramètre. Le processeur `xstlproc` a, par exemple, des options `--param` et `--stringparam` dont les valeurs sont des expressions XPath. La seconde option ajoute implicitement les apostrophes `'''` nécessaires autour des chaînes de caractères.

La feuille de style suivante utilise un paramètre global `bg-color` pour la couleur de fond du document XHTML résultat. Sa valeur est utilisée pour donner une régle CSS dans l'entête du document.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dbk="http://docbook.org/ns/docbook"
                xmlns="http://www.w3.org/1999/xhtml">
  <xsl:output ... />
  <!-- Paramètre global pour la couleur du fond -->
  <xsl:param name="bg-color" select="'white'"/>

  <xsl:template match="/">
    <xsl:comment>Generated by dbk2html.xsl</xsl:comment>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title><xsl:value-of select="dbk:book/dbk:title"/></title>
      <style>
        <xsl:comment>
          body { background-color: <xsl:value-of select="$bg-color"/>; }
        </xsl:comment>
      </style>
    </head>
    <body><xsl:apply-templates/></body>
    </html>
  </xsl:template>
  ...
```

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

Pour changer la couleur de fond du document résultat, il faut donner une autre valeur au paramètre `bg-color` comme dans l'exemple suivant.

```
xsltproc --stringparam bg-color blue dbk2html.xsl dbk2html.xml
```

8.9.2.2. Paramètres locaux

La déclaration d'un paramètre d'une règle est réalisée par un élément `xsl:param` enfant de `xsl:template`. Les déclarations de paramètres doivent être les premiers enfants. Le passage d'une valeur en paramètre est réalisé par un élément `xsl:with-param` fils de `xsl:apply-templates` ou `xsl:call-template`. Comme pour `xsl:variable`, l'attribut `name` détermine le nom de la variable. La valeur est donnée soit par une expression XPath dans l'attribut `select` soit directement dans le contenu de l'élément `xsl:variable`. Un attribut optionnel `as` peut spécifier le type de la valeur.

Dans l'exemple suivant, la première règle (pour la racine `'/'`) applique la règle pour le fils `text` avec le paramètre `color` égal à `blue`. La valeur par défaut de ce paramètre est `black`.

```
<?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" indent="yes"/>

  <xsl:template match="/">

    <xsl:apply-templates select="text">

      <!-- Valeur du paramètre pour l'appel -->

      <xsl:with-param name="color" select="'blue'"/>

    </xsl:apply-templates>

  </xsl:template>

  <xsl:template match="text">

    <!-- Déclaration du paramètre avec 'black' comme valeur par défaut --
>

    <xsl:param name="color" select="'black'"/>

    <p style="color:{$color};"><xsl:value-of select="."/></p>

  </xsl:template>

</xsl:stylesheet>
```

Dans l'exemple précédent, la valeur passée en paramètre est une chaîne de caractères mais elle peut aussi être un nœud ou une liste de nœuds. Dans l'exemple

suivant, la règle `get-id` retourne un attribut `id`. La valeur de celui-ci est produite à partir des attributs `id` et `xml:id` du nœud passé en paramètre. Par défaut, ce nœud est le nœud courant.

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet                                                version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

  <xsl:template match="/">

    <xsl:apply-templates select="*"/>

  </xsl:template>

  <xsl:template match="*">

    <xsl:copy>

      <!-- Appel de la règle get-id avec la valeur par défaut du
paramètre -->

      <xsl:call-template name="get-id"/>

      <xsl:apply-templates select="*"/>

    </xsl:copy>

  </xsl:template>

  <!-- Retourne un attribut id -->

  <xsl:template name="get-id">

    <!-- Paramètre avec le noeud courant comme valeur par défaut -->

    <xsl:param name="node" as="node()" select="."/>

    <xsl:attribute name="id"

                   select="($node/@id,      $node/@xml:id,      generate-
id($node))[1]"/>

  </xsl:template>

</xsl:stylesheet>
```

L'élément `xsl:param` est également utilisé pour déclarer les paramètres des fonctions d'extension XPath. La règle `get-id` de la feuille de style précédente aurait aussi pu être remplacée par une fonction d'extension. Dans la feuille de style suivante, la fonction XPath `get-id()` calcule la valeur de l'attribut `id` à partir des attributs `id` et `xml:id` du nœud passé en paramètre.

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="2.0"

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

              xmlns:xsd="http://www.w3.org/2001/XMLSchema"

                xmlns:fun="http://www.liafa.jussieur.fr/~carton">
```

```
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes"/>
  <!-- Définition de la fonction d'extension get-id -->
  <xsl:function name="fun:get-id" as="xsd:string">
    <xsl:param name="node" as="node()"/>
    <xsl:sequence select="($node/@id, $node/@xml:id, generate-id($node))[1]"/>
  </xsl:function>
  <xsl:template match="/">
    <xsl:apply-templates select="*"/>
  </xsl:template>
  <xsl:template match="*">
    <xsl:copy>
      <!-- Ajout de l'attribut id -->
      <xsl:attribute name="id" select="fun:get-id(.)"/>
      <xsl:apply-templates select="*"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

8.9.3. Récursivité

XPath 2.0 a ajouté des fonctions qui facilitent le traitement des chaînes de caractères. Avec XSLT 1.0 et XPath 1.0, il est souvent nécessaire d'avoir recours à des règles récursives pour certains traitements. Les deux règles`quote.string` et `quote.string.char` données ci-dessous insèrent un caractère `'\'` avant chaque caractère `'\'` ou `'"'` d'une chaîne.

```
<!-- Insertion de '\' avant chaque caractère '"' et '\' du contenu -->
<xsl:template match="text()">
  <xsl:call-template name="quote.string">
    <xsl:with-param name="string" select="."/>
  </xsl:call-template>
</xsl:template>
<!-- Insertion de '\' avant chaque caractère '"' et '\' du paramètre string -->
<xsl:template name="quote.string">
  <!-- Chaîne reçue en paramètre -->
  <xsl:param name="string"/>
  <xsl:choose>
```

```
    <xsl:when test="contains($string, '&quot;')">
      <xsl:call-template name="quote.string.char">
        <xsl:with-param name="string" select="$string"/>
        <xsl:with-param name="char"   select="'&quot;'"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($string, '\')">
      <xsl:call-template name="quote.string.char">
        <xsl:with-param name="string" select="$string"/>
        <xsl:with-param name="char"   select="'\'"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$string"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<!-- Fonction auxiliare pour quote.string -->
<xsl:template name="quote.string.char">
  <!-- Chaîne reçue en paramètre -->
  <xsl:param name="string"/>
  <!-- Caractère reçu en paramètre -->
  <xsl:param name="char"/>
  <xsl:variable name="prefix">
    <xsl:call-template name="quote.string">
      <xsl:with-param   name="string"   select="substring-before($string,
$char)"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:variable name="suffix">
    <xsl:call-template name="quote.string">
      <xsl:with-param   name="string"   select="substring-after($string,
$char)"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:value-of select="concat($prefix, '\', $char, $suffix)"/>
</xsl:template>
```

Ce traitement pourrait être réalisé de façon plus concise avec la fonction XPath 2.0 `replace()`.

```
<!-- Insertion de '\' avant chaque caractère '"' et '\' du contenu -->
<xsl:template match="text()">
  <xsl:value-of select="replace(., '([&quot;\\])', '\\$1')">
</xsl:template>
```

8.9.4. Paramètres tunnel

Il est parfois fastidieux de transmettre systématiquement des paramètres aux règles appliquées. Les *paramètres tunnel* sont transmis automatiquement. En revanche, ils ne peuvent être utilisées que dans les règles qui les déclarent.

Dans l'exemple suivant, la règle pour la racine applique une règle au nœud `text` avec les paramètres `tunnel` et `lunnet`. La règle appliquée reçoit ces deux paramètres et applique à nouveau des règles à ses fils textuels. Le paramètre `tunnel` est transmis implicitement à ces nouvelles règles car il a été déclaré avec l'attribut `tunnel` valant `yes`. La règle appliquée au nœuds textuels déclare les paramètres `tunnel` et `lunnet`. La valeur de`tunnel` est effectivement la valeur donnée au départ à ce paramètre. Au contraire, la valeur du paramètre `lunnet` est celle par défaut car il n'a pas été transmis.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                          version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="iso-8859-1"/>
  <xsl:template match="/">
    <xsl:apply-templates select="text">
      <xsl:with-param name="tunnel" select="'tunnel'" tunnel="yes"/>
      <xsl:with-param name="lunnet" select="'lunnet'"/>
    </xsl:apply-templates>
  </xsl:template>
  <xsl:template match="text">
    <xsl:apply-templates select="text()"/>
  </xsl:template>
  <xsl:template match="text()">
    <!-- L'attribut tunnel="yes" est nécessaire -->
    <xsl:param name="tunnel" select="'default'" tunnel="yes"/>
    <xsl:param name="lunnet" select="'default'"/>
    <!-- Produit la valeur 'tunnel' fournie au départ -->
```

```
    <xsl:value-of select="$tunnel"/>
    <!-- Produit la valeur 'default' déclarée par défaut -->
    <xsl:value-of select="$lunnet"/>
  </xsl:template>
</xsl:stylesheet>
```

8.10. Fonctions d'extension XPath

Avec XSLT 2.0, il est possible de définir des nouvelles fonctions qui peuvent être utilisées dans les expressions XPath. Ces nouvelles fonctions viennent complèter la librairie des fonctions XPath. Elles remplacent avantageusement des constructions lourdes de XSLT 1.0 avec des règles récursives. Le traitement des chaînes de caractères est un champ d'utilisation classique de ces fonctions.

La définition d'une fonction XPath est introduite par l'élément `xsl:function`. Cet élément a des attributs `name` et `as` qui donnent respectivement le nom et le type de retour de la fonction. Le nom de la fonction est un nom qualifié avec un espace de noms. Les paramètres de la fonction sont donnés par des éléments `xsl:param` enfants de l'élément `xsl:function`. Chacun de ces éléments a aussi des attributs `name` et `as` qui donnent respectivement le nom et le type du paramètre. En revanche, l'élément `xsl:param` ne peut pas donner une valeur par défaut au paramètre avec un attribut `select` ou un contenu. Cette restriction est justifiée par le fait que les fonctions XPath sont toujours appelées avec un nombre de valeurs correspondant à leur arité. Les types possibles pour le type de retour et les paramètres sont les types XPath. La définition d'une fonction prend donc la forme générique suivante.

```
<xsl:function name="name" as="return-type">
  <!-- Paramètres de la fonction -->
  <xsl:param name="param1" as="type1"/>
  <xsl:param name="param2" as="type2"/>
  ...
  <!-- Corps de la fonction -->
  ...
</xsl:function>
```

L'élément `xsl:function` doit nécessairement être enfant de l'élément racine `xsl:stylesheet` de la feuille de style. Ceci signifie que la portée de la définition d'une fonction est la feuille de style dans son intégralité.

La fonction `url:protocol` de l'exemple suivant extrait la partie protocole d'une URL. Elle a un paramètre `url` qui reçoit une chaîne de caractères. Elle retourne la

chaîne de caractères située avant le caractère `':'` si l'URL commence par un protocole ou la chaîne vide sinon.

```
<xsl:function name="url:protocol" as="xsd:string">
  <xsl:param name="url" as="xsd:string"/>
  <xsl:sequence select="
    if (contains($url, ':')) then substring-before($url, ':') else ''"/>
</xsl:function>
```

Une fois définie, la fonction `url:protocol` peut être utilisée comme n'importe quelle autre fonction XPath. L'exemple suivant crée un nœud texte contenant `http`.

```
<xsl:value-of select="url:protocol('http://www.liafa.jussieu.fr/')"/>
```

La fonction `url:protocol` peut être utilisée pour définir une nouvelle fonction `url:address` qui extrait la partie adresse internet d'une URL. Cette fonction utilise une variable locale `protocol` pour stocker le résultat de la fonction `url:protocol`.

```
<xsl:function name="url:address" as="xsd:string">
  <xsl:param name="url" as="xsd:string"/>
  <xsl:variable name="protocol" as="xsd:string"
select="url:protocol($url)"/>
  <xsl:sequence select="
    if (($protocol eq 'file') or ($protocol eq ''))
    then ''
    else substring-before(substring-after($url, '://'), '/')"/>
</xsl:function>
```

L'expression XPath `url:address('http://www.liafa.jussieu.fr/~carton/')` s'évalue, par exemple, en la chaîne de caractères `www.liafa.jussieu.fr`.

Les fonctions définies par `xsl:function` peuvent bien sûr être récursives. La fonction récursive suivante `url:file` extrait le nom du fichier d'un chemin d'accès. C'est la chaîne de caractères située après la dernière occurrence du caractère `'/'`.

```
<xsl:function name="url:file" as="xsd:string">
  <xsl:param name="path" as="xsd:string"/>
  <xsl:sequence select="
    if (contains($path, '/'))
```

```
    then url:file(substring-after($path, '/'))
    else $path"/>
</xsl:function>
```

L'expression XPath `url:file('Enseignement/XML/index.html')` s'évalue, par exemple, en la chaîne de caractères `index.html`.

Une fonction XPath est identifiée par son nom qualifié et son arité (nombre de paramètres). Il est ainsi possible d'avoir deux fonctions de même nom pouvu qu'elles soient d'arités différentes. Ceci permet de simuler des paramètres avec des valeurs par défaut en donnant plusieurs définitions d'une fonction avec des nombres de paramètres différents. Dans l'exemple suivant, la fonction `fun:join-path` est définie une première fois avec trois paramètres. La seconde définition avec seulement deux paramètres permet d'omettre le troisième paramètre qui devient ainsi optionnel.

```
<!-- Définition d'une fonction join-path avec 3 paramètres -->
<xsl:function name="fun:join-path" as="xsd:string">
  <xsl:param name="path1" as="xsd:string"/>
  <xsl:param name="path2" as="xsd:string"/>
  <xsl:param name="sep"   as="xsd:string"/>
  <xsl:sequence select="concat($path1, $sep, $path2)"/>
</xsl:function>
<!-- Définition d'une fonction join-path avec 2 paramètres -->
<xsl:function name="fun:join-path" as="xsd:string">
  <xsl:param name="path1" as="xsd:string"/>
  <xsl:param name="path2" as="xsd:string"/>
  <xsl:sequence select="concat($path1, '/' , $path2)"/>
</xsl:function>
...
  <!-- Appel de la fonction à 3 paramètres -->
  <xsl:value-of select="fun:join-path('Directory', 'index.html', '/')"/>
  <!-- Appel de la fonction à 2 paramètres -->
  <xsl:value-of select="fun:join-path('Directory', 'index.html')"/>
```

8.11. Modes

Il est fréquent qu'une feuille de style traite plusieurs fois les mêmes nœuds du document d'entrée pour en extraire divers fragments. Ces différents traitements peuvent être distingués par des *modes*. Chaque règle de la feuille de style déclare

pour quel mode elle s'applique avec l'attribut `mode` de l'élément `xsl:template`. En parallèle, chaque application de règles avec `xsl:apply-templates` spécifie un mode avec un attribut `mode`.

Chaque mode est identifié par un identificateur. Il existe en outre les valeurs particulières `#default`, `#all` et `#current` qui peuvent apparaître dans les valeurs des attributs `mode`.

La valeur de l'attribut `mode` de l'élément `xsl:template` est soit la valeur `#all` soit une liste de modes, y compris `#default`, séparés par des espaces. La valeur `#current` n'a pas de sens dans ce contexte et ne peut pas apparaître. La valeur par défaut est bien sûr `#default`.

```
<!-- Règle applicable avec le mode #default -->
<xsl:template match="...">
...
<!-- Règle applicable avec le mode test -->
<xsl:template match="..." mode="test">
...
<!-- Règle applicable avec les modes #default foo et bar -->
<xsl:template match="..." mode="#default foo bar">
...
<!-- Règle applicable avec tous les modes -->
<xsl:template match="..." mode="#all">
...
```

La valeur de l'attribut `mode` de l'élément `xsl:apply-templates` est soit `#default` soit `#current` soit le nom d'un seul mode. La valeur `#all` n'a pas de sens dans ce contexte et ne peut pas apparaître. La valeur `#current` permet appliquer des règles avec le même mode que celui de la règle en cours. La valeur par défaut est encore `#default`.

```
<!-- Application avec le mode #default -->
<xsl:apply-templates select="..."/>
...
<!-- Application avec le mode test -->
<xsl:apply-templates select="..." mode="test"/>
...
<!-- Application avec le mode déjà en cours -->
<xsl:apply-templates select="..." mode="#current"/>
```

```
...
```

Dans l'exemple suivant, le nœud `text` est traité d'abord dans le mode par défaut `#default` puis dans le mode `test`. Dans chacun de ces traitements, ses fils textuels sont traités d'abord dans le mode par défaut puis dans son propre mode de traitement. Les fils textuels sont donc, au final, traités trois fois dans le mode par défaut et une fois dans le mode `test`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                                version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="iso-8859-1"/>
  <xsl:template match="/">
    <xsl:apply-templates select="text"/>
    <xsl:apply-templates select="text" mode="test"/>
  </xsl:template>
  <xsl:template match="text" mode="#default test">
    <xsl:apply-templates select="text()"/>
    <xsl:apply-templates select="text()" mode="#current"/>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:text>Mode #default: </xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>
  <xsl:template match="text()" mode="test">
    <xsl:text>Mode test: </xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

L'exemple suivant illustre une utilisation classique des modes. Le document est traité une première fois en mode `toc` pour en extraire une table des matières et une seconde fois pour créer le corps du document proprement dit.

```
<!-- Règle pour la racine -->
<xsl:template match="/">
  <html>
```

```
    <head>
      <title><xsl:value-of select="book/title"/></title>
    </head>
    <body>
      <!-- Fabrication de la table des matières -->
      <xsl:apply-templates mode="toc"/>
      <!-- Fabrication du corps du document -->
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
...
<!-- Règles pour la table des matières -->
<xsl:template match="book" mode="toc">
  <h1>Table des matières</h1>
  <ul><xsl:apply-templates mode="toc"/></ul>
</xsl:template>
```

8.12. Indexation

La fonction `id()` permet de retrouver des nœuds dans un document à partir de leur attribut de type `ID`. Elle prend en paramètre une liste de noms séparés par des espaces. Elle retourne une liste contenant les éléments dont la valeur de l'attribut de type `ID` est un des noms passés en paramètre. Cette fonction est typiquement utilisée pour traiter des attributs de type `IDREF` ou `IDREFS`. Ces attributs servent à référencer des éléments du document en donnant une (pour `IDREF`) ou plusieurs (pour `IDREFS`) valeurs d'attibuts de type `ID`. La fonction `id()` permet justement de retrouver ces éléments référencés.

L'exemple suivant illustre l'utilisation classique de la fonction `id()`. On suppose avoir un document contenant une bibliographie où les éditeurs ont été placés dans une section séparée du document. Chaque élément `book`contient un élément `published` ayant un attribut `by` de type `IDREF` pour identifier l'élément `publisher` correspondant dans la liste des édideurs. La feuille de style suivante permet de revenir à une bibliographie où chaque élément `book` contient directement l'élément `publisher` correspondant. L'élément retourné par la fonction `id()` est stocké dans une variable pour l'utiliser à plusieurs reprises.

```
<?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" indent="yes"/>
  <!-- Règle pour la racine -->
  <xsl:template match="/">
    <bibliography>
      <xsl:apply-templates select="bibliography/books/book"/>
    </bibliography>
  </xsl:template>
  <!-- Règles pour les livres -->
  <xsl:template match="book">
    <book>
      <!-- Copie des attributs -->
      <xsl:copy-of select="@*"/>
      <!-- Copie des éléments autres que published -->
      <xsl:copy-of select="*[name() != 'published']"/>
      <!-- Remplacement des éléments published -->
      <xsl:apply-templates select="published"/>
    </book>
  </xsl:template>
  <!-- Règle pour remplacer published par le publisher référencé -->
  <xsl:template match="published">
    <!-- Elément publisher référencé par l'élément published -->
    <xsl:variable name="pubnode" select="id(@by)"/>
    <publisher>
      <!-- Recopie des attributs autres que id -->
      <!-- L'attribut id ne doit pas être recopié car sinon,
           on peut obtenir plusieurs éléments publisher avec
           la même valeur de cet attribut -->
      <xsl:copy-of select="$pubnode/@*[name() != 'id']"/>
      <xsl:copy-of select="$pubnode/* | $pubnode/text()"/>
    </publisher>
  </xsl:template>
</xsl:stylesheet>
```

Lorsque la fonction `id()` retourne plusieurs nœuds, il est possible de les traiter un par un en utilisant un élément `xsl:for-each` comme dans l'exemple suivant.

```
  <xsl:for-each select="id(@areafefs)">
    <xsl:apply-templates select="."/>
```

```
  </xsl:for-each>
```

Afin de pouvoir accéder efficacement à des nœuds d'un document XML, il est possible de créer des index. L'élément `xsl:key` crée un index. L'élément `xsl:key` doit être un enfant de l'élément racine `xsl:stylesheet`. L'attribut `name` de `xsl:key` fixe le nom de l'index pour son utilisation. L'attribut `match` contient un motif qui détermine les nœuds indexés. L'attribut `use` contient une expression XPath qui spécifie la clé d'indexation, c'est-à-dire la valeur qui identifie les nœuds et permet de les retrouver. Pour chaque nœud sélectionné par le motif, cette expression est évaluée en prenant le nœud sélectionné comme nœud courant. Pour indexer des éléments en fonction de la valeur de leur attribut `type`, on utilise l'expression `@type` comme valeur de l'attribut `use`. La valeur de l'attribut `use` peut être une expression plus complexe qui sélectionne des enfants et des attributs. Dans l'exemple suivant, tous les éléments `chapter` du document sont indexés en fonction de leur attribut `id`.

```
<xsl:key name="idchapter" match="chapter" use="@id"/>
```

La fonction `key()` de XPath permet de retrouver un nœud en utilisant un index créé par `xsl:key`. Le premier paramètre est le nom de l'index et le second est la valeur de la clé. Dans l'exemple suivant, on utilise l'index créé à l'exemple précédent. La valeur de l'attribut `@idref` d'un élément contenu dans la variable `$node` sert pour retrouver l'élément dont c'est la valeur de l'attribut `id`. Le nom `idchapter` de l'index est un nom XML et il doit être placé entre apostrophes ou guillemets.

```
<xsl:value-of select="key('idchapter', $node/@idref)/title"/>
```

L'utilisation des index peut être contournée par des expressions XPath appropriée. L'expression ci-dessus avec la fonction `key` est équivalente à l'expression XPath ci-dessous qui utilise l'opérateur `'//'`.

```
<xsl:value-of select="//chapter[@id = $node/@idref]/title"/>
```

L'inconvénient de cette dernière expression est d'imposer un parcours complet du document pour chacune de ses évaluations. Si l'expression est évaluée à de nombreuses reprises, ceci peut conduire à des problèmes d'efficacité.

Lors de la présentation des attributs de type `ID` et `IDREF`, il a été observé que la bibliographie `bibliography.xml` doit être organisée de façon différente si les éléments `publisher` contiennent des informations autres que le nom de l'éditeur. Afin d'éviter de dupliquer ces informations dans chaque livre, il est préférable de regrouper les éléments `publisher` dans une section à part. Chaque élément `publisher` contenu dans un élément `book`est alors remplacé par un élément `published` avec un attribut `by` de type `IDREF` pour référencer

l'élément `publisher` déplacé. La feuille de style suivante remplace les éléments `publisher` par des éléments `published`et les regroupe dans une liste en fin de document. Elle suppose que chaque élément `publisher` contient, au moins, un enfant `name` avec le nom de l'éditeur. Elle supprime également les doublons en évitant que deux éléments `publisher` avec le même nom, c'est-à-dire le même contenu de l'élément `name`, apparaissent dans la liste. Cette suppression des doublons est réalisée par une indexation des éléments `publisher` sur le contenu de `name`. Pour chaque élément `publisher`, l'indexation permet de retrouver efficacement tous les éléments `publisher` avec un contenu identique de `name`. Seul le premier de ces éléments `publisher` est ajouté à la liste des éditeurs dans le document résultat.

Pour savoir si un élément `publisher` est le premier de ces éléments avec le même nom, celui-ci est comparé avec le premier de la liste retournée par la fonction `key()` avec l'opérateur `is`. La copie de cet élément`publisher` est réalisée par un élément `xsl:copy` ainsi que par élement `xsl:copy-of` pour ses attributs et ses enfants.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                          version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="iso-8859-1" indent="yes"/>
  <!-- Indexation des éléments publisher par leur nom -->
  <xsl:key name="pubname" match="publisher" use="name"/>
  <xsl:template match="/">
    <bibliography>
      <books>
        <!-- Livres -->
        <xsl:apply-templates select="bibliography/book"/>
      </books>
      <publishers>
        <!-- Éditeurs -->
        <xsl:apply-templates select="bibliography/book/publisher"/>
      </publishers>
    </bibliography>
  </xsl:template>
  <!-- Règles pour les livres -->
  <xsl:template match="book">
    <book>
      <!-- Copie des attributs et des enfants autres que publisher -->
```

```
      <xsl:copy-of select="@* | *[name() != 'publisher']"/>
      <!-- Transformation de l'élément publisher en élément published -->
      <xsl:if test="publisher">
        <published                          id="{generate-id(key('pubname',
publisher/name)[1])}"/>
      </xsl:if>
    </book>
  </xsl:template>
  <!-- Copie d'un élément publisher en ajoutant un attribut id -->
  <xsl:template match="publisher">
   <!-- Test si l'élément publisher est le premier avec le même nom -->
    <xsl:if test=". is key('pubname', name)[1]">
      <xsl:copy>
        <!-- Ajout de l'attribut id -->
        <xsl:attribute name="id">
          <xsl:value-of select="generate-id()"/>
        </xsl:attribute>
        <!-- Copie des attributs et des enfants -->
        <xsl:copy-of select="@* | *"/>
      </xsl:copy>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

8.13. Documents multiples

La fonction `document()` permet de lire et de manipuler un document XML contenu dans un autre fichier. Le nom du fichier contenant le document est fourni en paramètre à la fonction. Le résultat peut être stocké dans une variable ou être utilisé directement. Le document suivant référence deux autres documents `europa.xml` et `states.xml`.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<files>
  <file href="europa.xml"/>
  <file href="states.xml"/>
</files>
```

Le document `europa.xml` est le suivant. Le document `states.xml` est similaire avec des villes américaines.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<cities>
  <city>Berlin</city>
  <city>Paris</city>
</cities>
```

La feuille de style XSL suivante permet de collecter les différentes villes des documents référencés par le premier document pour en faire une liste unique.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet                                          version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
  <html>
    <head>
      <title>Liste de villes</title>
    </head>
    <body>
      <h1>Liste de villes</h1>
      <ul>
        <xsl:for-each select="files/file">
          <xsl:apply-templates select="document(@href)/cities/city"/>
        </xsl:for-each>
      </ul>
    </body>
  </html>
  </xsl:template>
  <xsl:template match="city">
    <li><xsl:value-of select="."/></li>
  </xsl:template>
</xsl:stylesheet>
```

L'ajout d'un élément `xsl:sort` comme fils de l'élément `<xsl:apply-templates select="document(@href)/cities/city"/>` permet de trier les éléments à l'intérieur d'un des documents référencés. Pour trier les éléments globalement, il faut

supprimer les deux itérations imbriquées et les remplacer par une seule itération comme dans l'exemple suivant.

```
<xsl:apply-templates select="document(files/file/@href)/cities/city">
  <xsl:sort select="."/>
</xsl:apply-templates>
```

Dans l'expression XPath `document(files/file/@href)/cities/city`, on utilise le fait que la fonction `document()` prend en paramètre une liste de noms de fichiers et qu'elle retourne l'ensemble des contenus des fichiers.

8.14. Analyse de chaînes

Il arrive que le document source ne soit pas suffisament structuré et que la feuille de style ait besoin d'extraire des morceaux de texte. L'élément `xsl:analyze-string` permet d'analyser une chaîne de caractères et de la découper en fragments. Ces fragments peuvent ensuite être repris et utilisés. L'analyse est réalisée avec une expression rationnelle.

La chaîne à analyser et l'expression rationnelle sont respectivement données par les attributs `select` et `regex` de l'élément `xsl:analyze-string`. Les deux enfants `xsl:matching-substring` et `xsl:non-matching-substring` de `xsl:analyze-string` donnent le résultat suivant que la chaîne est compatible ou non avec l'expression.

La fonction XPath `regex-group()` permet de récupérer un fragment de la chaîne correspondant à un bloc délimité par des parenthèses dans l'expression. L'entier fourni en paramètre donne le numéro du bloc. Les blocs sont numérotés à partir de 1.

Dans l'exemple suivant, le contenu de l'élément `name` est découpé à la virgule pour extraire le prénom et le nom de famille d'un nom complet écrit suivant la convention anglaise. Les deux parties du nom sont ensuite utilisées pour construire les enfants `firstname` et `lastname` de `name`. Lorsque le contenu ne contient pas de virgule, l'élément `name` est laissé inchangé.

```
<xsl:template match="name">
  <xsl:analyze-string select="." regex="([^,]*),\s*(.*)">
    <xsl:matching-substring>
      <name>
        <firstname><xsl:value-of select="regex-group(2)"/></firstname>
        <lastname><xsl:value-of  select="regex-group(1)"/></lastname>
      </name>
```

```
    </xsl:matching-substring>
    <xsl:non-matching-substring>
      <name><xsl:value-of select="."/></name>
    </xsl:non-matching-substring>
  </xsl:analyze-string>
</xsl:template>
```

La feuille de style précédente peut être appliquée au document suivant.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<names>
  <name>Lagaffe, Gaston</name>
  <name>Talon, Achille</name>
  <name>Astérix</name>
</names>
```

On obtient le document suivant où les noms sont mieux structurés.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<names>
   <name><firstname>Gaston</firstname><lastname>Lagaffe</lastname></name>
   <name><firstname>Achille</firstname><lastname>Talon</lastname></name>
   <name>Astérix</name>
</names>
```

8.15. Import de feuilles de style

Les éléments `xsl:include` et `xsl:import` permettent d'inclure les règles d'une feuille de style au sein d'une autre feuille de style. La seule différence entre les deux éléments est la gestion des priorités entre les règles des deux feuilles de style. Ces priorités influencent les choix de règles à appliquer sur les nœuds. Ces deux éléments ont un attribut `href` contenant l'URL de la feuille de style à inclure. Cette URL est très souvent le nom relatif ou absolu d'un fichier local. Les deux éléments `xsl:include` et `xsl:import` doivent être enfants de l'élément racine `xsl:stylesheet` et ils doivent être placés avant toute définition de règle par un élément`xsl:template`. L'exemple suivant est la feuille de style principale pour la transformation de cet ouvrage au format DocBook en HTML.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
```

```
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dbk="http://docbook.org/ns/docbook">
  <!-- Import de la feuille de style docbook.xsl -->
  <xsl:import
    href="/usr/share/sgml/docbook/xsl-stylesheets/html/profile-
docbook.xsl"/>
  <!-- Ajout des paramètres communs de configuration -->
  <xsl:include href="common.xsl"/>
  <!-- Feuille de style CSS -->
  <xsl:param name="html.stylesheet" select="'style.css'"/>
  <!-- Pour les paragraphes justifiés à droite -->
  <xsl:template match="dbk:para[@role = 'right']">
    <p align="right"><xsl:apply-templates/></p>
  </xsl:template>
</xsl:stylesheet>
```

Lorsque un processeur XSLT choisit une règle à appliquer à un nœud du document source, il prend en compte deux paramètres qui sont la *priorité d'import* entre les feuilles de style et les priorités entre les règles. Lorsque la feuille de style est importée avec l'élément `xsl:include`, les deux feuilles de style ont même priorité d'import comme si les règles des deux feuilles se trouvaient dans une même feuille de style. Au contraire, lorsque la feuille de style est importée avec l'élément `xsl:import`, la feuille de style importée a une priorité d'import inférieure. Les règles de la feuille de style qui réalise l'import, c'est-à-dire celle qui contient l'élément `xsl:import`, sont appliquées en priorité sur les règles de la feuille importée. Ce mécanisme permet d'adapter la feuille de style importée en ajoutant des règles dans la feuille de style qui importe. Lorsqu'une règle ajoutée remplace une règles de la feuiile importée, elle peut encore utiliser la règle remplacée grâce à l'élément `xsl:apply-imports`.

Une feuille de style peut importer plusieurs feuilles de style avec plusieurs éléments `xsl:import`. Dans ce cas, les feuilles de style importées en premier ont une priorité d'import inférieure. L'ordre des priorités d'import est l'ordre des éléments `xsl:import` dans la feuille de style qui importe. Il est également possible qu'une feuille de style importée par un élément `xsl:import` réalise elle-même des imports d'autres feuilles de style avec des éléments `xsl:import`. Lorsqu'il y a plusieurs niveaux d'import, l'ordre d'import entre les différentes feuilles de style est d'abord dicté par l'ordre des imports de premier niveau puis l'ordre des imports de second niveau et ainsi de suite. Ce principe est illustré par l'exemple suivant. Supposons qu'une feuille de style A importe, dans cet ordre, les feuilles de style B et C, que la feuille B importe, à son tour, les feuilles D et E et que la feuille C importe, finalement, les feuilles F et G (cf. Figure). L'ordre des feuilles par priorité

d'import croissante est D, E, B, F, G, C, A. Cet ordre correspond à un parcours suffixe de l'arbre des imports.

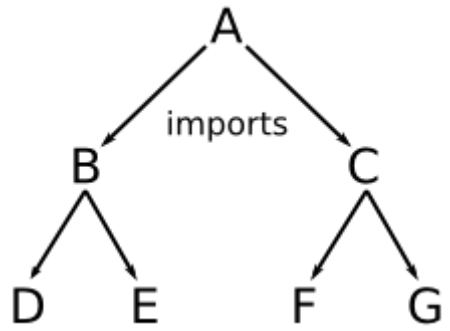

Figure 8.4. Priorités d'import