Auteur

Ferey Cyril

Version

Version	*Date*	*Objet*
1	31/08/04	Document initial
1.1	04/04/2006	Ajout de la notion de priorité dans le traitement des nœuds XSLT

Sources

Cours XML	Ferey Cyril
Cours MSXML3	Ferey Cyril
Spécification du format XML Nomade	Ferey Cyril
Liste des Expression Xpath	VincentQuint
SelfHTML	http://fr.selfhtml.org/xml/index.htm
Exercices	Ferey Cyril & Différentes participations à des forums

A noter : ce cours a été écrit librement sur mon temps libre, il est fournit gratuitement, et en aucun cas cela ne retire de droits à son auteur.

Diffusion

08/12/2008

Organisme	*Destinataire*	*Pour Action*	*Pour Info*
Web	www.nanokrill.com		x

COURS

Introduction

Le but de ce cours n'est pas d'énoncer l'ensemble des solutions possibles du xslt, mais de permettre de poser des ancres afin de vous aider dans la recherche de vos solutions.

Généralités

eXtensible Stylesheet Language Transformation : partie de XSL, ensemble de règles de transformation d'un document XML vers un autre document.

Les exemples ci dessous se base sur les dernière normes du WC3, elles sont donc compatible avec les parsers suivant
MSXML 3 & >
Jaxp 1.1
Saxon
Si vous voulez installé l'un de ces parser allez en annexe « Parser » pour consulter quelques sites et indications

Mais avant d'entrer dans l'utilisation du XSLT, il faut commencer par bien former son XML

Document XML

Généralités

XML (eXtensible Markup Language) est un "langage de balisage extensible", développé par la W3C (http://w3.org). "Langage de balisage Extensible" signifie que le XML n'est pas limité (tout comme le HTML) à un balisage prédéfini extrêmement figé mais au contraire tout un chacun pourra définir ses propres balises.

Exemple : <telephone>00.00.00.02.01</telephone>
est une ligne comprise par le XML et exploitable, alors qu'elle ne le sera jamais en HTML

Au cours de ces dernières années le langage HTML à énormément évolué, et avec cette évolution sont apparues de nombreuses balises, auxquelles ont du s'adapter les navigateurs du marché. Mais l'HTML restant essentiellement un langage de présentation, ces nouveaux apports ont rendu de plus en plus complexes les fichiers HTML ainsi que leurs traitements par les navigateurs du marché : ceci rendant leur exécution (affichage) beaucoup moins rapide et efficace.

L'HTML atteint aujourd'hui certaines limites et le W3C s'est attaché à développer une nouvelle norme "le XML" qui n'est pas fondamentalement (le HTML à sans doute encore de belles années devant lui) un changement mais plus une adaptation aux situations nouvelles (Ex. les applications de e-commerce de plus en plus évoluées, messageries, ...)

Le XML cherche avant tout à dissocier les données de leur mise en forme. Le HTML lui aussi a évolué dans ce sens en introduisant depuis quelques années la notion de feuilles de style, mais ceci n'a rien à voir avec la dissociation beaucoup plus structurées faîtes par le XML. "Le XML ne cherche pas à afficher les éléments mais à les stocker"

Notations

Balise texte simple ou nœud

<element>valeur de l'élément</element>

Element Vide

On désigne en XML "l'élément vide" tout élément dont le contenu ("valeur de l'élément") est vide.
Notation :
<elementvide></elementvide>
ou
<elementvide />

Exemple d'éléments vides :

<img src="toto.gif"></img> est un élément vide
ou encore
<hreference ul="http://www.asp-magazine.com" />

Le noms des éléments

Les noms des éléments doivent se soumettre à plusieurs règles :
là encore les noms sont souvent une source d'erreur.

1- Respecter la casse
la notation <Adresse></ADRESSE> est incorrecte, en effet le XML est sensible à la casse. Pour corriger la notation employée précédément, on écrira :
<adresse></adresse>
Par convention, on écrira le nom des éléments toujours en minuscule.

2- Les caractères non autorisés
<telephone personnel></telephone personnel> est incorrect en XML
on écrira :
<telephone-personnel></telephone-personnel>

3- Les caractères autorisés
La ponctuation : "-" , ":" , "_" , "."
Les caractères alphabétiques accentués ou pas : "a" , "A" , "é" , etc...

Les caractères numériques : "1", "2", "3" etc..

4- La première lettre du nom
- Le nom d'un élément commence toujours par une lettre ou un des caractères de ponctuation suivant : "-", ":","_","."
Note : Même s'il est possible de l'utiliser, évitez l'emploi du ":" (deux points) car il est réservé pour les espaces de noms.

Exemple :
<toto></toto> est correct
<1toto></1toto> est incorrect

·5- Les noms commençant par "xml" ou par n'importe quel caractères "X" ou "x", "M" ou "m", "l" ou "L" sont à éviter car ils sont réservés à des fins de normalisation du langage.
Pour respecter cette norme, vous devez vous fixer des conventions au départ :
Par exemple :
Je remplacerais l'espace par un tiret : <tel-personnel>
ou je mettrais une majuscule a chaque début de nouveau mot : <TelPersonnel>
ou autre ...

Le tout étant d'éviter des mélanges de plusieurs conventions au sein d'un même fichier , et de rester compatible avec les normes à venir ...
La racine du document
Tout document XML doit contenir une racine : Un élément parent qui "chapeaute" l'ensemble des autres éléments du document.

Exemple :
```
<?xml version "1.0"?>
<news>
<titre>Valider une adresse email</titre>
<date-parution>17/11/00</date-parution>
<auteur />
</news>
<news>
<titre>Envoyer un fichier par mail</titre>
<date-parution>10/11/00</date-parution>
<auteur />
</news>
```

est incorrect
La version correcte du code précédent, est la définition d'une racine qui chapeaute l'ensemble du document :

```
<?xml version "1.0"?>
<aspmag-news>
```

<news>
<titre>Valider une adresse email</titre>
<date-parution>17/11/00</date-parution>
<auteur />
</news>
<news>
<titre>Envoyer un fichier par mail</titre>
<date-parution>10/11/00</date-parution>
<auteur />
</news>
</aspmag-news>
est correct.

Dans l'exemple précédent aspmag-news est l'élément parent du document.
Il ne peut exister qu'un élément parent par document.
Les commentaires
Tout langage qui se respecte, dispose d'une norme de nommage pour les commentaires : les commentaires sont toujours ignorés des compilateurs, et sont simplement placés par le programmeur à "titre de documentation du code".
Notation :
<!-- Ceci est un commentaire -->

Le codage des caractères : Unicode

L'écriture des caractères XML se conforme à la norme UNICODE, norme développée afin de facilité l'internationalisation des fichiers écrits.
http://www.unicode.com

Nous utiliserons dans nos fichiers XML la norme ISO 8859-1qui correspond aux langues d'Europe occidentale (Latin-1), ce qui permet d'utiliser la plupart des accentuations possibles.

Déclaration de fichier XML écrit en Latin-1 :
```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Les entités

Ce sont des références pointant vers des caractères ou des groupes de caractères (Le concept des entités est relativement complexe, nous y reviendrons plus tard).
Les entités sont comprises entre le & (et commercial) et le ; (point virgule).

Quelques entités prédéfinies :

< remplace le <
> remplace le >
& remplace le caractère espace
' remplace le caractère ' (apostrophe)
" remplace le caractère " (guillemet)

On peut aussi remplacer un caractères en utilisant une entité décimale (&#---;) ou son code hexadécimal (&#x--;)
Exemple : î

Les sections CDATA

Même si les entités peuvent servir à remplacer un certain nombre de caractères (par exemple le < ou le >) dans un texte, vous vous rendrez rapidement compte qu'en pratique cette notation peut rendre compliqué un texte (essayer par exemple d'utiliser les entités dans un script javascript : c'est illisible .. non ?).

Une section CDATA permet de référencer un texte pouvant contenir n'importe quel caractère (y compris les caractères de balisage habituellement réservé). L'ensemble des balises pouvant être contenu dans une section CDATA est ignorée du processeur XML.

Notation :

```
<[CDATA[ ...
Texte .... <balise>texte balise </balise>
Dans ce texte les balises sont ignorées du processeur ...
]]>
```

Les attributs

Nous avons vu qu'un élément XML se définit entre des balises :
`<element>Valeur de l'element</element>`

En plus de sa valeur , il est possible d'associer des informations supplémentaires à un élément : un attribut.

Notation :

```
<element attribut="valeur de l attribut">Valeur de l'élément</attribut>
```

Exemple :

```
<personne sexe="masculin">DUPONT</personne>
```

Les attributs spéciaux

`xml:lang` : cet attribut permet de définir la langue dans lequel est écrit le fichier XML
`xml:space` : cet attribut permet permet d'exclure ou non les espaces dupliqués (doubles espaces) dans un nom

Espace de nommage

Un document XML est constitué essentiellement d'éléments et d'attributs. Par la référence à une DTD, ces éléments et attributs appartiennent à l'espace de nommage de cette DTD. Si la déclaration de type de document manque cependant et si le document est certes conforme, mais non valide, alors il n'y a pas d'appartenance claire. Il reste incertain d'"où" (de quel espace de nommage) viennent les noms d'attributs et d'éléments utilisés. À cela, il reste la possibilité de mentionner explicitement un espace de nommage pour un nom d'élément ou un nom d'attribut.

La désignation de l'espace de nommage est particulièrement importante quand les noms d'éléments ou d'attributs provenant d'espaces de nommage différents se contredisent. Supposons qu'il y ait deux fois, dans un document XML un élément nommé div. Une fois, il se réfère à un

espace de nommage distinct et une fois il doit remplir le rôle d'un élément HTML. Ce n'est que par la référence à un espace de nommage déterminé qu'il devient clair dans ce cas dans quel "contexte" l'élément doit être interprété.

à cette fin, le consortium W3 a introduit le concept des noms qualifiés (qualified names). Les noms qualifiés comprennent toujours un préfixe, qui désigne l'espace de nommage et une partie locale du nom, qui désigne le nom de l'élément ou de l'attribut dans l'espace de nommage. En travaillant simultanément avec plusieurs espaces de nommage, il est important de noter des noms qualifiés.

Vous pouvez définir dans un document XML des "îlots" avec des données de certains espaces de nommage.

Exemple:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<livre xmlns="http://www.monserveur.fr/XML/livre">
<chapitre numero="1">
  <html xmlns="http://www.w3.org/TR/REC-html-40">
    <head><titre>Introduction</titre></head>
    <body>
     <h1>Introduction</h1>
     <p>Le livre commence par ce texte...</p>
    </body>
   </html>
</chapitre>
</livre>
```

L'exemple montre un document XML. Il contient un élément document nommé livre. Dans son repère d'ouverture est contenue une déclaration d'espace de nommage XML. Pour ce faire, l'attribut xmlns= est noté dans le repère d'ouverture (xmlns = XML name space, donc espace de nommage XML). Derrière une URI suit immédiatement; elle mentionne à quel autre espace de nommage il sera fait référence dans cet élément. Il n'est pas indispensable que cette URI soit une adresse pouvant être appelée. Il s'agit d'une pure convention comparable à une attribution de nom sans ambiguïté. Pour vos propres langages XML, vous pouvez attribuer ce nom vous-même. Dans l'exemple l'adresse http://www.monserveur.fr/XML/livre a été choisie. l'élément livre proprement-dit et ses éléments subordonnés (éléments-enfants) - par exemple l'élément chapitre - se réfèrent maintenant à l'espace de nommage défini. Sous l'élément chapitre de l'exemple il y a pourtant à nouveau un élément avec déclaration d'espace de nommage: un élément html avec l'attribut xmlns. En ce qui concerne HTML 4.0 le consortium W3 prévoit pour le faire la mention http://www.w3.org/TR/REC-html-40. Tous les éléments enfants qui sont maintenant placés entre <html> et </html> font partie de l'espace de nommage HTML (ou bien: de l'ilôt de données HTML).

Utiliser simultanément plusieurs espaces de nommage

En travaillant avec des îlots de données les espaces de nommage sont séparés proprement les uns des autres par la hiérarchie des éléments. Il est toutefois également possible de mélanger des espaces de nommage. Pour ce faire, il vous faut noter les mentions de préfixe pour affecter à chaque fois l'espace de nommage auquel vous pensez.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<commande xmlns:produit="http://localhost/XML/produit"
xmlns:client="http://localhost/XML/client">
 <produit:numero>p49393</produit:numero>
 <produit:nom>JXY Rasierer VC100</produit:nom>
 <produit:quantite>1</produit:quantite>
 <produit:prix>69,--</produit:prix>
 <client:numero>c2029</client:numero>
 <client:nom>Marius, Escartefigues</client:nom>
 <client:adresselivraison>Cours Mirabeau 14, 13100 Aix en Provence</client:adresselivraison>
</commande>
```

Dans l'exemple, il s'agit d'une structure de commande. Une commande typique accède à deux bases de données différentes: pour le produit commandé à la base de données des produits en stock et pour celui qui commande à la base de données clients. C'est ce qui est exprimé dans l'exemple de la façon suivante: dans le repère d'ouverture de l'élément commande sont placées deux déclarations d'espaces de nommage: la première pour produit et la deuxième pour client. Entre <commande> et </commande> on doit pouvoir travailler avec les éléments des deux espaces de nommage. À cet effet des préfixes ont été définis dans les déclarations des espaces de nommage. À la place d'un simple attribut xmlns= on a noté ici xmlns:produit= et xmlns:client=, donc xmlns, suivi de deux points et d'un nom (le tout sans espace qui sépare) Les noms comme dans l'exemple produit et client peuvent être choisis librement. Comme attribution de valeur à l'attribut ainsi défini suit par convention une URI prescrite. Dans l'exemple http://localhost/XML/produit et http://localhost/XML/client ont été choisies.

Pour les éléments placés à l'intérieur de <commande> et </commande> , il est fait référence, grâce au préfixe défini, à l'espace de nommage désiré. produit:numero fait par exemple référence à l'espace de nommage qui a été défini avec xmlns:produit="http://localhost/XML/produit", alors que client:numero fait référence à l'espace de nommage qui a été défini avec xmlns:client="http://localhost/XML/client".

Exploitation des données

Bien que ce cours soit orienté vers l'utilisation des transformations (xslt), Il faut savoir qu'il existe plusieurs technique pour stocker en mémoire un document XML afin de travailler dessus.
DOM
SAX

Seule le DOM est utilisé en interne pour le moment, il permet d'avoir à un instant t le document entier en mémoire et de faire des traitements sur l'ensemble des données, son utilisation est réservée à de petite quantités de données (inférieur à 1 Mo pour des performances optimale sur les frontaux).
A l'inverse dans le cas de traitement de données de back end (de 3 Mo à 1Go) il faut utilise la technique Sax qui permet de ne charger en mémoire qu'une partie des données.

DOM (Document Object Model)

est une API orientée objet permettant d'accéder aux fichiers XML en créant un arbre logique dont chaque élément est un nœud de l'arbre.

SAX (Simple API for XML)

est une API orientée évènement (Ex. le fait de charger une page est un évènement ONLOAD) permettant de la même manière d'exploiter des fichiers XML. Le concept évènement sera dans certaines situations beaucoup plus efficace que l'approche objet, en effet alors que DOM stocke une copie entière de son arbre en mémoire, SAX ne transmettra les éléments que s'il y a lieu de le faire.

TRANSFORMATION DU XML

Introduction

Nous avons vu que nous allons utiliser la technique du DOM avec un document XML entièrement stocké en mémoire. Il s'agit alors d'un arbre que nous allons parcourir

Parcours de l'arbre

XSLT décrit des règles pour transformer un arbre source représentant le document xml en un arbre résultat. Cette transformation s'effectue par rapport à des modèles (templates) décrits dans la feuille de style. La construction de l'arbre de résultat permet de réordonner ou de filtrer le contenu du fichier xml. La feuille de style contient un ensemble de règles de modèles, chaque règle a deux parties : un motif dont on cherche la correspondance avec les noeuds de l'arbre source et un modèle qui sera utilisé pour former l'arbre résultat.

Appel de la feuille de style dans le document XML

La feuille de style XSL est enregistrée dans un fichier externe et son nom comporte l'extension ".xsl". Dans le document XML, l'instruction de traitement suivante indique le type de la feuille de style et son emplacement :
<?xml-stylesheet type="text/xml" href="URL"?>

Prenons l'exemple d'une bibliothèque, chaque livre a un titre, un auteur et une référence, le document XML ci-dessous appelle une feuille de style XSL qui est dans le fichier biblio.xsl dans le répertoire courant :

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

<bibliotheque>

<livre>
        <titre>N ou M</titre>
        <auteur>Agatha Christie</auteur>
        <ref>Policier-C-15</ref>
</livre>

<livre>
        <titre>Le chien des Baskerville</titre>
        <auteur>Sir Arthur Conan Doyle</auteur>
        <ref>Policier-D-3</ref>
</livre>

<livre>
        <titre>Dune</titre>
        <auteur>Franck Heckbert</auteur>
        <ref>Fiction-H-1</ref>
</livre>

</bibliotheque>
```

Structure d'une feuille de style

XSL est une application XML, une feuille de style XSL est donc un document XML. La feuille de style contient donc une déclaration XML et tous ses éléments sont placés dans l'élément racine. D'autre part, les éléments XSL sont préfixés par xsl: (XSL utilise les domaines de noms).

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

L'élément racine contient principalement des modèles (templates) pour l'affichage du document XML.

Modèle unique pour un document XML

Une feuille de styles XSL contient un ou plusieurs modèles (templates), chaque modèle contient des informations sur l'affichage d'une branche des éléments du document. S'il n'y a qu'un seul modèle alors il s'applique sur la racine du document XML.

Prenons la feuille de style biblio.xsl qui applique un modèle unique au document XML bibliothèque :

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

<xsl:template match="/">

 <html>
   <head>
     <title>Ma bibliotheque</title>
   </head>
   <body>
         <H2>Bibliotheque</H2>

         <xsl:for-each select="bibliotheque/livre">
         <SPAN style="font-style:italic; padding-right:3pt">
             <xsl:value-of select="titre"/>
         </SPAN>
         <SPAN style="color:red; padding-right:3pt">
             <xsl:value-of select="auteur"/>
         </SPAN>
         <SPAN style="color:blue">
             <xsl:value-of select="ref"/>
         </SPAN>
         <br />
         </xsl:for-each>

    </body>
    </html>

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

Cette feuille de style comporte une déclaration XML, un élément racine xsl:stylesheet qui englobe tous les autres éléments et précise que les éléments préfixés par xsl: appartiennent au domaine de nom xsl.

Le modèle est appliqué à la branche spécifiée par l'attribut match de l'élément template. La transformation d'un document XML par une feuille de style XSL s'effectue donc par un modèle traitant un noeud donné. Chaque modèle est divisé en deux parties : un noeud cible indiqué par l'attribut match et une action sur le noeud :

<xsl:template match="noeud_cible">
action
</xsl:template>

Bonnes méthodes

Il faut distinguer deux choses dans un document XSLT, le parcour de l'arbre et les éléments de présentation (xhtml ou autre). Nous avons vu qu'il est possible pour un nœud donné d'appliquer un template. Mais il

vaux mieux différencier dans le xslt le code du pour de l'arbre et le code de présentation, en tout cas pour les xml plus complexe.
Donc en premier le parcour de l'arbre et la modification des embranchements

PARCOURS DE L'ARBRE

Modification du parcours de l'arbre

La façon la plus simple d'imaginer le résultat étant de se dire que l'arbre xml est parcouru dans son intégralité par défaut et que à chaque fois que le programme rencontre un motif (match ou expression Xpath) correspondant à l'arbre ou le nœud qu'il est en train de parcourir, il va déclencher les actions que se trouvent dans votre template.

```
<xsl:template match="pattern">
action
</xsl:template>
```

pattern : motif, pattern, ou Expression Xpath définissant les cibles de la règles.
Exemple match='film' va se déclencher pour chaque noeud film.

Mise en forme d'éléments cible à partir d'un nœud existant

```
<xsl:template match="pattern">
        actions
        <xsl:apply-templates select=''cible''>
</xsl:template>

<xsl:template match= 'cible'>
action
</xsl:template>
```

Ici lorsque l'arbre contient la première pattern il déclenchera les actions du premier template et demandera l'application du deuxième template sur le nœud « cible ».

Utilisation de plusieur template sur le même nœud
Il est possible de prévoir plusieurs template sur le même nœud en ajoutant dans la définition et dans l'appel de celui-ci un mode particulier

Utilisation d'un template comme fonction
L'utilisation de ce principe est très pratique pour séparer le parcours de l'arbre de l'affichage du template.

```
<xsl:call-template name="affChannel">
      <xsl:with-param name="nbElement" select="count(siteMapNode)"/>
```

```
</xsl:call-template>

<xsl:template name="affChannel">
        <xsl:param name="nbElement"></xsl:param>
</xsl:template>
```

Passage de valeurs entre templates

```
 <!—parcours de l'arbre→
<xsl:template match="pattern">
        actions
        <xsl:apply-templates select=”cible”>
</xsl:template>

<xsl:template match= ‘cible’>
actions
        <xsl:apply-templates select="//siteMapNode[@id=$recursifref]">
                        <xsl:with-param name="parentId" select="$parentId"/>
        </xsl:apply-templates>
</xsl:template>

<xsl:template match="siteMapNode">
        <xsl:param name="parentId"></xsl:param>
        actions
</xsl:template>
```

Résumé des attributs

Attribut Description
match="pattern" traite les noeuds sélectionnés par le pattern spécifié, au lieu de traiter tous les éléments.
name="nom" traite les noeuds sélectionnés par le pattern spécifié, au lieu de traiter tous les éléments.
priority="niveau" applique les templates avec un mode donné.
mode="mode" applique les templates avec un mode donné.

Priorité sur l'application d'un template

Il arrive souvent que plusieurs modèles sélectionnent le même élément dans le fichier XML source. Il convient donc de décider lequel doit être utilisé. Cet ordre de priorité peut être spécifié à l'aide de l'attribut priority. Si ce dernier n'est pas présent, le calcul de la priorité s'effectue selon plusieurs règles
La priorité du modèle CCC est plus faible que celle de CCC/CCC, car il est moins précis.
La priorité du modèle CCC est plus faible que CCC/CCC et AAA/CCC/CCC, mais ces deux derniers modèles ont la même priorité. Dans ce cas, un processeur XSLT peut signaler l'erreur. Toutefois, s'il ne signale pas d'erreur, il doit décider quelle est la dernière règle modèle dans la feuille de style parmi les règles modèle concordantes restantes.

la règle modèle la moins précise "*" comporte une priorité inférieure à celle de la règle CCC. Le calcul des priorités s'échelonne de -0.5 à 0.5. La spécification XSLT fournit plus de précisions.

CONDITIONS ET FONCTIONS DANS LE PARCOUR DE L'ARBRE

Généralité

Nous avons vu tout ce qui concerne les templates, maintenant voyons ce que l'on peut en faire

Contenu du Template

L'élément <xsl:template> accepte en son sein les éléments suivants :

xsl:apply-templates
xsl:attribute
xsl:call-template
xsl:choose
xsl:comment
xsl:copy
xsl:copy-of
xsl:element
xsl:for-each
xsl:if
xsl:param
xsl:processing-instruction
xsl:text
xsl:value-of
xsl:variable
éléments HTML

Apply-template/call-template
Nous avons déjà vu que cette instruction permet de se brancher sur un autre template, elle fait partie intégrante du parcours de l'arbre, les autres instructions permettent elle de modifie les éléments de sortie.

Attribute
L'élément <xsl:attribute> insère un attribut avec le nom indiqué dans l'arborescence d'un document résultant.

```
<xsl:attribute name="nom_attribut">
   Valeur de l'attribut
</xsl:attribute>
```

L'attribut name permet d'affecter un nom à l'attribut créé. Evidemment cette commande est obligatoire.

Exemple
<img>

```
        <xsl :attribute name= ‘src’>
                <xsl :value-of select= ‘image/@src’>
        </xsl :attribute>
</img>
```
donne en sortie :

Choose
L'élément <xsl:choose> combiné avec <xsl:when> et <xsl:otherwise>, permet de construire des tests conditionnels à l'instar des commandes switch de Java ou Javascript.

```
<xsl:choose>
  <xsl:when test="condition">
    instructions...
  </xsl:when>
  ...
  <xsl:otherwise>
    instructions...
  </xsl:otherwise>
</xsl:choose>
```

Comment
L'élément <xsl:comment> permet d'insérer un commentaire dans l'arborescence d'un document XML.

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

Copy
L'élément <xsl:copy> est utilisé afin de recopier le noeud courant dans l'arborescence du document XML résultant.

```
<xsl:copy>
  template...
</xsl:copy>
```

Copy-of
L'élément <xsl:copy-of> permet de recopier les noeuds sélectionnés par le pattern dans l'arbre du document XML résultant.

<xsl:copy-of select="pattern"/>Les attributs :
Elément Description
select="pattern" permet de sélectionner par une expression des noeuds.

Element
L'élément <xsl:element> permet d'insérer d'un marqueur dans l'arborescence d'un document XML résultant.

<xsl:element
name="nom_élément"
namespace="adresse_URI"
use-attribute-sets="nom_jeu_attributs">
...
</xsl:element>Les attributs :
Elément Description
name="nom" permet de donner un nom à l'élément XML créé.
namespace="adresse_URI" permet de spécifier l'adresse URI d'un espace de noms.

For-each
L'élément <xsl:for-each> permet d'appliquer des règles de style sur chaque noeud identique d'un template.

Les noeuds sont identifiés par un pattern spécifié par un attribut select d'ailleurs obligatoire.

<xsl:for-each select="pattern">
instructions...
</xsl:for-each>Les attributs :
Attribut Description
select="pattern" permet de sélectionner une série de noeuds dans un template.

Evidemment, cet élément ne peut être employé que sur une arborescence dont la structure soit uniforme et connue tel que par exemple un document XML formé à partir de titres et de paragraphes.

If
L'élément <xsl:if> permet d'appliquer un test conditionnel dans la structure d'une feuille de style XSL.

<xsl:if test="condition">
Instructions...
</xsl:if>Si le test conditionnel est vérifié alors le processeur XSL exécutera les instructions contenues à l'intérieur des marqueurs, sinon il les ignorera et passera aux instructions suivantes.

test="condition" permet de poser une condition d'éxécution.

Param
L'élément <xsl:param> permet de déclarer un paramètre utilisable par une règle de modèle <xsl:template>.

<xsl:param name="nom">
Valeur...
</xsl:param>La valeur du paramètre se trouve au sein des marqueurs.

Si l'attribut select est employé, alors l'élément <xsl:param> devra être un élément vide.

<xsl:param name="nom" select="expression"/>

PI
L'élément <xsl:processing-instruction> permet de créer une instruction de traîtement dans un document XML résultant.

<xsl:processing-instruction name="nom">
  Instructions...
</xsl:processing-instruction>Les instructions sont en fait des attributs avec leurs valeurs qui apparaîtront dans l'instruction de traîtement.

<?nom_instruction attributs="valeur"?>Les attributs :
Elément Description
name="nom" affecte un nom à l'instruction de traîtement.

Texte
L'élément <xsl:text> permet d'insérer des données textuelles dans un document XML résultant.

<xsl:text disable-output-escaping="yes|no">
  Données textuelles...
</xsl:text>Les attributs :
Attribut Description
disable-output-escaping="yes|no" active ou désactive le remplacement des caractères spéciaux par leur entité XML.
A noter que cet élément peut être utilisé en combinaison à un CDATA

Value-of
L'élément <xsl:value-of> permet d'extraire la valeur d'un noeud sélectionné dans l'arborescence d'un document XML.

<xsl:value-of
  select="expression"
  disable-output-escaping="yes|no"/>Cet élément ne peut traiter qu'un seul noeud à la fois contrairement à <xsl:apply-tempates> qui gére un ensemble de noeuds sélectionné par son pattern.

Utilisé en conjonction avec l'instruction <xsl:for-each>, l'élément <xsl:value-of> peut alors traiter une série de noeuds correspondant à l'expression de l'attribut select.

Les attributs :
Attribut Description
select="pattern" sélectionne des noeuds dans une arborescence source.
disable-output-escaping="yes|no" active ou désactive le remplacement des caractères spéciaux par leur entité XML.

L'élément <xsl:value-of> peut non-seulement sélectionner des éléments XML, mais aussi des attributs par l'intermédiaire du signe @ ainsi que des paramètres ou des variables avec $.

L'élément précitée ne peut être contenu que dans une règle de modèle <xsl:template> ou un appel à cette dernière <xsl:call-template>.

Variable
L'élément <xsl:variable> permet de déclarer une variable dans une feuille de style.

<xsl:variable name="nom">
  Valeur...
</xsl:variable>La valeur du paramètre se trouve au sein des marqueurs.

Si l'attribut select est utilisé, alors <xsl:variable> doit être un élément vide et le contenu de l'attribut sera la valeur du paramètre.

<xsl:variable name="nom" select="expression"/>Les attributs :
Attribut Description
name="nom" affecte un nom au paramètre.
select="pattern" sélectionne un type de noeuds dans l'arborescence du document XML.

Dans le cas où la variable est déclarée dans l'élément <xsl:stylesheet>, alors elle serait valable dans l'ensemble de la feuille.

Déclarée dans une règle de modèle <xsl:template>, la variable a une portée locale.

Une variable est appelée par l'intermédiaire de son nom qui doit être précédé du signe dollar ($).

Deux éléments <xsl:variable> ne peuvent se faire mutuellement référence.

Elements Html
Les éléments de sortie HTML permettent de présenter clairement des données XML dans un navigateur web.

Ce type de transformation est non seulement très utile, mais fait également preuve d'une quasi-parfaite portabilité sur des systèmes d'exploitation ou sur des navigateurs.

Toutes les balises HTML peuvent être employées au sein des éléments de tranformation XSL.

Néanmoins, le balisage HTML doit correspondre à celui de la version 4.0 du langage et ainsi être parfaitement structuré.

Toutes les balises doivent être fermées, y compris les éléments vides tels que <img> ou
.

Les marqueurs doivent s'imbriquer correctement.

Malgré l'absence d'une méthode de transformation annoncée par l'élément <xsl:output>,

le processeur XSL active automatiquement une sortie sous forme HTML, si la feuille de style de transformation contient le marqueur <html> avec n'importe quelle combinaison de casse.

XPATH

Expression Xpath

L'expression Xpath peut être utilisée dans

- attribut match du template
- attribut select de value-of , for-each
- attribute test de if
-

Une expression XPath

- s'évalue en fonction d'un noeud contexte
- désigne un ou plusieurs chemins dans l'arbre à partir du noeud contexte
- a pour résultat
  - un ensemble de noeuds
  - ou une valeur, numérique, booléenne ou alphanumérique

L'expression XPATH est constitué d'un ensemble de paramétres optionnels

Location path

La plus importante construction du langage Xpath

Un location path est une expression XPath qui retourne un node-set
Un location path est formé d'une suite de location steps séparés par des '/'
Un location step est évalué par rapport au noeud courant et retourne un node-set dont chaque noeud devient le noeud courant pour l'évaluation du step suivant

Deux types de location path
Relatif : commence au noeud courant (contexte d'évaluation)
Absolu : commence au noeud racine du document contenant le noeud courant indiqué par un '/' initial

Location step

Un location step a trois parties
1. un axe : relation structurale entre le noeud courant et les noeuds retournés par le step
2. un test : type et nom des noeuds retournés par le step
3. des prédicats optionnels qui ravinent la sélection opérée par le step

Les prédicats sont appliqués l'un après l'autre, après la sélection par l'axe et le test
Chaque prédicat retourne un node-set filtré par le prédicat suivant
Syntaxe d'un location step :

axe::test[prédicat][prédicat]
Exemple :
child::para[position()=1]

Axe
L'axe d'un location step indique dans quel partie de l'arbre (par rapport au noeud courant) il faut chercher les noeuds à retourner :

child : tous les enfants du noeud courant
descendant : enfants du noeud courant, enfants des enfants, etc.
parent : parent du noeud courant
ancestor : parent du noeud courant, parent du parent, etc.
following-sibling : tous les frères suivants du noeud courant
preceding-sibling : tous les frères précédents du noeud courant
following : tous les noeuds qui sont après le noeud courant (pas les descendants)
preceding : tous les noeuds qui sont avant le noeud courant (pas les ancêtres)
attribute : les attributs du noeud courant
namespace : les noeuds espace de noms du noeud courant
self : le noeud courant seul
descendant-or-self : le noeud courant et ses descendants
ancestor-or-self : le noeud courant et ses ancêtres

Test
Le test d'un location step indique quels noeuds (type et/ou nom) il faut choisir sur l'axe considéré
Les axes concernent des types de noeuds différents :

attribute : noeuds attribut
namespace : noeuds espace de noms
autres axes : noeuds éléments

Formes des tests :
nom (QName) : noeuds du type correspondant à l'axe et qui ont ce nom
exemples : child::para, attribute::href
* : noeuds correspondant à l'axe, quel que soit leur nom
exemples : child::*, attribute::*
text() : noeuds de type texte
exemple : child::text()
comment() : noeuds de type commentaire
processing-instruction() : noeuds de type PI
un argument permet d'indiquer le nom de la PI concernée

Prédicat
Un prédicat filtre un *node–set* en fonction de l'axe pour produire un nouveau *node–set*
Un prédicat est une expression booléenne évaluée en fonction du contexte :
- noeud courant

- taille : nombre de noeuds dans le *node–set* filtré
- position : rang du noeud dans le *node–set*, dans l'ordre du document (ou dans l'ordre inverse pour les axes ancestor, ancestor–or–self, preceding, et preceding–sibling)

Chaque noeud du *node–set* n'est conservé que si l'évaluation du prédicat pour ce noeud retourne *true*
Exemple de prédicats :
last()
position() mod 2 = 1
count() < 5

Syntaxe abrégée

child:: est l'axe par défaut : il peut être omis
exemple : div/para est équivalent à child::div/child::para

@ est l'abréviation de attribute::
exemple : para[@type="warning"] est équivalent à child::para[attribute::type="warning"]

// est l'abréviation de /descendant-or-self::node()/
exemple : //para

. est l'abréviation de self::node()
exemple : .//para est équivalent à self::node()/descendant-or-self::node()/child::para

.. est l'abréviation de parent::node()
exemple : ../title est équivalent à parent::node()/child::title

Expressions XPath
Les location paths ne sont pas les seuls types d'expressions XPath
Il y a aussi des expressions booléennes, numériques et chaînes de caractères
Toute expression XPath peut être utilisée dans un prédicat de location path
Dans une expression XPath on peut utiliser un location path, ou une union de location paths séparés par 'l'

Booléens
On peut combiner plusieurs expressions booléennes élémentaires par les opérateurs and et or
Les expressions élémentaires sont des comparaisons :
= != < > <= >=
Les comparaisons se font après avoir converti les objets à comparer en un même type
La comparaison d'un *node–set* avec une chaîne se fait en cherchant un noeud du *node–set* dont la *string–value* est la même que la chaîne
Précédence des opérateurs :

or
and
=, !=
<=, <, >=, >

On peut utiliser les opérateurs
+ – div mod
Si les opérandes ne sont pas des nombres, ils sont convertis en nombres : les *node–sets* sont convertis en chaînes, elles–mêmes converties en nombres

Fonctions
Toute implémentation de XPath doit comprendre au moins les fonctions de base (Core Function Library)
Les fonctions utilisables dans l'évaluation d'une expression sont définies par le contexte d'évaluation de l'expression
Les arguments de la fonctions sont convertis si nécessaire pour être du type voulu (booléen, nombre, chaîne), sauf les argument du type *node–set* qui ne peuvent être convertis
La conversion se fait après appel des fonctions boolean, number, string

Fonctions node–sets
last() : un nombre, la taille du contexte
position() : un nombre, la position du contexte
count(*node–set*) : nombre de noeuds dans le *node–set*
id(*object*) : *node–set* contenant les éléments du document qui ont pour ID la valeur de *object*
local–name(*node–set*) : la chaîne du nom local du premier noeud de *node–set*
namespace–uri(*node–set*) : la chaîne de l'URI de l'espace de nom du premier noeud de *node–set*
name(*node–set*) : la chaîne du nom complet du premier noeud de *node–set*

Fonctions chaînes
string(*object*) : convertit *object* en une chaîne de caractères
concat(*string1*, *string2*, *string**) : concatène plusieurs chaînes
start–with(*string*1, *string2*) : booléen, vrai si *string1* commence par *string2*
contains(*string*1, *string*2) : booléen, vrai si *string1* contient *string2*
substring–before(*string1*, *string2*) : la sous–chaîne de *string1* qui précède la première occurrence de *string2*
substring–after(*string1*, *string2*) : la sous–chaîne de *string1* qui suit la première occurrence de *string2*
substring(*string*, *number1*, *number2*) : la sous–chaîne de *string* qui commence à la position *number1* et dont la longueur est *number2*

string–length(*string*) : nombre de caractères dans *string*
normalize–space(*string*) : retire les espaces initiaux, finals et les espaces doublés

Fonctions booléennes
boolean(*object*) : convertit *object* en booléen
true si nombre non nul, *node–set* non vide, chaîne de longueur non nulle
not(*boolean*) : négation de *boolean*
true()
false()
lang(*string*) : la langue (attribut xml:lang) du noeud courant est la même ou un sous–langage de *string*

Fonctions numériques
number(*object*) : convertit *object* en un nombre
sum(*node–set*) : somme du résultat de la conversion en nombre de chaque noeud de *node–set*
floor(*number*) : plus grand entier inférieur ou égal à *number*
ceiling(*number*) : plus petit entier supérieur ou égal à *number*
round(*number*) : entier le plus proche de *number*

Exemples

Pour les exemples ci dessous nous nous baserons sur l'arbre suivant

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
   < !—Com1→
   <a>
      <b att1="b1">
         Texte B1
         <d>D1</d>
      </b>
      <b  att2="b2">Texte B2</b>
      <c att2="c1">
         Texte C1
         <d>D2</d>
      </c>
   </a>
</document>
```

Exemple de XSL appliqué

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

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="EXPRESSION">
    <xsl:value-of select="."/>
</xsl:template>
</xsl:stylesheet>
```

EXPRESSION REPREND LES VALEURS SUIVANTES
En fonction de l'utilisation de l'expression (value-of, template match) le renvoi est une référence sur un nœud ou la valeur de celui. Il faut noter que dans le DOM une valeur est considéré comme un nœud terminal (tout est nœud).

Expression		Renvoi
/a/b/[@att1]	:	b1

Dans l'expression `/A/B[@att1]` :

On s'intéresse aux noeuds de type `B` fils de l'élément racine `A`. Parmi ces noeuds on ne prend que ceux pour lesquels le prédicat `[@att1]` s'évalue à `true` Cette expression s'évalue avec pour noeud contexte un élément `b[@att1]` vaut `true` ssi `@att1` renvoie un ensemble de noeuds **non vide**

//@att2	:	C1
attribute ::att2	:	C1
/a/*	:	nœuds b,b,c
/a/b/d	:	noeuds dans b premier et c
//descendant ::text()	:	peu utilisé sous cette forme, renvoi les textes de l'ensemble des nœuds
/comment()	:	Com1

EXERCICES

Tous les exercices se font avec le XML suivant

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<articles>
    <article>
        <id>FI015166</id>
        <datesortie>2004-09-08</datesortie>
        <titreoriginal>DIARIOS DE MOTOCICLETA</titreoriginal>
        <titre>CARNETS DE VOYAGE</titre>
        <histoire>En 1952, deux jeunes argentins, Alberto Granado et Ernesto Guevara, décident de partir à la découverte de leur continent : l'Amérique latine. Ils commencent leur expédition sur une vieille moto Norton 500 de 1939, baptisée  la Poderosa (la Puissante)...</histoire>
        <imagewap height="96" width="75">http://media.monsieurcinema.com/film/015100/15166/iw15166.jpg</imagewap>
        <note>3</note>
        <films>
            <film id="FI009606">
                <titre_film><![CDATA[CENTRAL DO BRASIL]]></titre_film>
            </film>
            <film id="FI014748">
                <titre_film><![CDATA[ATTRACTION FATALE]]></titre_film>
            </film>
        </films>
    </article>
    <article>
```

```
        <id>FI015255</id>
        <datesortie>2004-09-09</datesortie>
        <titreoriginal>LAWS OF ATTRACTION</titreoriginal>
        <titre/>
        <histoire>Audrey Woods, avocate spécialisée dans les divorces, rencontre un autre spécialiste de ce genre d'affaires, Daniel
Rafferty. Entre eux s'installe un climat de rivalité mais se développe également une attraction mutuelle qui va compliquer inévitablement
leur travail.</histoire>
        <imagewap height="96" width="72">http://media.monsieurcinema.com/film/015200/15255/iw15255.jpg</imagewap>
        <note>4</note>
        <films>
            <film id="FI009606">
                <titre_film><![CDATA[CENTRAL DO BRASIL]]></titre_film>
            </film>
            <film id="FI010030">
                <titre_film><![CDATA[PILE ET FACE]]></titre_film>
            </film>
        </films>
    </article>
</articles>
```

Afficher l'ensemble des titres de films

Afficher les titres de films dont le film a une note supérieure à 3

Afficher l'ensemble des titres de film faisant référence au film CENTRAL DO BRASIL

Si vous avez des difficultés à faire ces exercices, vous trouverez en annexe d'autres exercices XML/XSLT plus linéaire avec leur correction. Ceux ci dessus sont fait pour être corrigé en groupe.

EXEMPLES D'APPLICATIONS

TODO

QUESTIONS COURANTES

Si j'applique deux règle sur le même nœud que se passe t'il ?
C'est la régle (expression) la plus précise qui est prioritaire.
Exemple

```
<xsl :template match='a'/>
        actions
```

```
</xsl:template>
<xsl :template match=’a[position()=1]’/>
        actions
</xsl:template>
```

En l’absence d’autres instruction c’est le template 2 qui sera exécuté.

Comment tester la présence d’un nœud ?

Par un test boolean sur celui-ci la fonction normalize permet de supprimer les espace à droite et à gauche de la chaîne de caractère ce qui évite d’afficher l’élément si celui-ci ne contient que des espaces

```
<xsl:when test="boolean(normalize-space(@id))"><xsl:value-of select="$Id"/></xsl:when>
```

Je ne veux afficher que le premier article, comment faire ?

Il faut appliquer le template uniquement sur le premier article

Exemple

```
<xsl:template match="/">
 <xsl:apply-templates select="//article[1]"/>
</xsl:template>
```

Je ne veux afficher que les 3 premiers articles ?

Il faut appliquer le template sur les article dont la position est inférieure à 3

```
<xsl:template match="/">
        <xsl:apply-templates select="//article[position()&lt;=3]"/>
</xsl:template>
```

Puis-je mettre un script dans mon XSL ?

Si celui-ci est fait pour apparaître sur la page de sortie, je conseille l’utilisation d’une balise Texte et CDATA qui permet de mettre du contenu brut à l’intérieur.

Je dois traiter un XML entier qui n’est pas classé, dois-je faire un classement sur chaque appel de template ?

Non il est possible de recréer un xm complet dans une variable et d’appliquer sur chacun des éléments de cette variable les même fct que sur le xml entier

XSL

```
    <xsl:variable name="ordered_produits">
        <root>
            <xsl:apply-templates select="//Cat[@id = $ChoiceCatid]/Asso/Produit[(@visible = '1') and (@visualisation = 'externe') and
(@sorte = 1) and (Editeur/@visible = '1')]" mode="ordered">
                <xsl:sort select="@position" data-type="number"/>
            </xsl:apply-templates>

            <xsl:apply-templates select="//Cat[@id = $ChoiceCatid]/Asso/Produit[(@visible = '1') and (@visualisation = 'externe') and
(@sorte != 1) and (Editeur/@visible = '1')]" mode="ordered">
                <xsl:sort select="@position" data-type="number"/>
            </xsl:apply-templates>
        </root>
      </xsl:variable>
```

Utilisation du nouvel XML

```
<xsl:apply-templates select="msxsl:node-set($ordered_produits)//root/newproduit" />
```

Je dois faire un traitement récursif sur une arborescence d'articles en limitant le niveau de traitement à 3 sur une arborescence en comptant 10, comment faire ?
Il faut que sur chaque traitement vous passier au traitement suivant l 'indice plus un. Indice représentant le niveau de l'arborescence et est initialisé à 1 au départ.

Est-il possible d'utiliser un langage plus évolué à l'intérieur d'un fichier XSL ?
Oui pour le parser Microsoft (Jscript, VbScript), c'est cependant déconseillé pour des raisons de compatibilité.
Il faut

- définir un namespace (xmlns:myfunc="http://myfunc")
- Ecrire la fonction

```
<msxsl:script language="JScript" implements-prefix="myfunc"><![CDATA[
function getWeekNr()
{ etc etc...
```

- Faire un appel à celle-ci

```
<xsl:variable name="AffSelection1" select="myfunc:GetSelectionNr(number($NbSelection), 1)" />
```

Je ne veux pas de barre de séparation sur mon dernier article ?
Il faut test si l'article est le dernier

```
<xsl:if test="position()!=last()">
    <hr width="228" color="#E5E5E5"/>
</xsl:if>
```

Je veux mettre une valeur en attribute, y'a plus simple que <attribute> ?
Oui il est plus simple d'utiliser { VALEUR} qui est l'équivalent dans un attribut de value-of

```
<a href="{@url}" target="_blank"><xsl:value-of select="@title"/></a>
```

Lorsque j'enregistre mon fichier XML j'ai l'erreur « entitie expected » ?
Cette erreur est généralement due à l'oublie d'encodage de & en &amp ; dans les liens.

Je veux mettre la première lettre d'une chaîne de caractère en majuscule, comment faire ?
Il faut coupe la première lettre de la chaîne et la transformer à l'aide de l'instruction
XML

```
<a>hours hours</a>
```

XSL

```
<xsl:value-of select="translate(substring-after(.,' '),'OURSINTE','oursinte')"/>
```

donne : hours HOURS

Comment faire pour devenir chef ?
Ce n'est pas possible en XSLT, cela demande d'autres compétences.

Comment faire un tri par date ?
Le type date n'existe pas en XSLT, il faut donc découper celle-ci et faire un tri sur chaque élément

Fichier XML

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

```
<articles>
    <article>
        <titre>Dolly</titre>
        <dateevenement>03/09/04</dateevenement>
        <corps>Parc des expositions de Langolvas - Morlaix (29)&lt;br/&gt;24€</corps>
    </article>
    <article>
        <titre>Alpha Blondy</titre>
        <dateevenement>04/09/04</dateevenement>
        <corps>Parc des expositions de Langolvas - Morlaix (29)&lt;br/&gt;25€</corps>
    </article>
    <article>
        <titre>Lynda Lemay</titre>
        <dateevenement>25/09/04</dateevenement>
        <corps>Casino de Paris - 20h30 - Paris (75)&lt;br/&gt;De 35€ à 53€</corps>
    </article>
</article>
```

XSL

```
<xsl:apply-templates select="//article">
        <xsl:sort select="substring(dateevenement, 7, 2)" data-type="number" order="ascending"/>
        <xsl:sort select="substring(dateevenement, 4, 2)" data-type="number" order="ascending"/>
        <xsl:sort select="substring(dateevenement, 1, 2)" data-type="number" order="ascending"/>
</xsl:apply-templates>
```

Appel du template article en classant les noeuds en function de la date

Comment répartir des résultat sur 2 colonnes ?

Plusieurs possibilités

1] La première, on doit afficher un élément une fois à droite une fois à gauche.

Solution : On parcourt deux fois les éléments

la première fois si la position (position()) de l'article est pair ((number(position()) mod 2=1) on l'affiche

la deuxième fois si la position de l'article est impair on l'affiche.

Fichier XSL

```
<xsl:template match="/">
        <!-- Le maximum d'article est 6 affichable, dans le cas contraire on laisse le nombre exact pour le calcul -->
         <xsl:variable name="NbConcerts">
             <xsl:if test="count(//article)>6">6</xsl:if>
             <xsl:if test="6>count(//article)"><xsl:value-of select="count(//article)"/></xsl:if>
         </xsl:variable>
         <!-- Formule permettant d'avoir une division entière dans le premier cas, et d'ajouter le restant de la première dans le second cas
-->
         <xsl:variable name="NbColonne2" select="$NbConcerts div 2 - (($NbConcerts mod 2) div 2)"/>
         <xsl:variable name="NbColonne1" select="$NbConcerts div 2 - (($NbConcerts mod 2) div 2 ) + $NbConcerts mod 2"/>
          <table class="contenu" border="0" cellpadding="0" cellspacing="0">
                <tr>

                <td width="230" valign="top">
                    <!-- element pair -->
                    <xsl:apply-templates select="//article">
                        <xsl:sort select="substring(dateevenement, 7, 2)" data-type="number" order="ascending"/>

                        <xsl:sort select="substring(dateevenement, 4, 2)" data-type="number" order="ascending"/>
                        <xsl:sort select="substring(dateevenement, 1, 2)" data-type="number" order="ascending"/>
                        <xsl:with-param name="bascule">1</xsl:with-param>
                    </xsl:apply-templates>
                </td>

                <td width="8"><img src="img/traitgv.gif"/></td>
```

```
                <td width="230" valign="top">
                    <!-- element impair -->
                    <xsl:apply-templates select="//article">
                        <xsl:sort select="substring(dateevenement, 7, 2)" data-type="number" order="ascending"/>

                        <xsl:sort select="substring(dateevenement, 4, 2)" data-type="number" order="ascending"/>
                        <xsl:sort select="substring(dateevenement, 1, 2)" data-type="number" order="ascending"/>
                        <xsl:with-param name="bascule">0</xsl:with-param>
                    </xsl:apply-templates>
                </td>
            </tr>
    </table>
</xsl:template>

<!-- template intermediaire permettant d'avoir des résultats classés-->
<xsl:template match="article">
    <!--<xsl:param name="bascule"/>-->
    <xsl:param name="AffichageBas"/>
    <xsl:param name="AffichageHaut"/>
    <!--<xsl:if test=" (number(position()) mod 2) =$bascule and position()&lt;=6">-->
    <xsl:if test="position()>$AffichageBas and position()&lt;=$AffichageHaut and position()&lt;=6">
        <xsl:call-template name="article"/>
    </xsl:if>
    <!--</xsl:if> -->
</xsl:template>


<xsl:template name="article">
        <span class="date"><xsl:value-of select="dateevenement" disable-output-escaping="yes"/></span> <a
href="concerts.aspx?id={position()}"><xsl:value-of select="titre" disable-output-escaping="yes"/></a> <br/>
</xsl:template>

</xsl:stylesheet>
```

2] On doit d'abord afficher l'ensemble des elements à droite puis à gauche.
Dans ce cas on calcule le nombre d'élément dans chaque colonne. Pour la deuxième, on divise le nombre total par deux, pour la première on reprend le même résultat et on ajoute le reste de la division entière (1 ou 0).

```
    <!-- Formule permettant d'avoir une division entière dans le premier cas, et d'ajouter le restant de la première dans le second cas -->
    <xsl:variable name="NbColonne2" select="$NbConcerts div 2 - (($NbConcerts mod 2) div 2)"/>
    <xsl:variable name="NbColonne1" select="$NbConcerts div 2 - (($NbConcerts mod 2) div 2 ) + $NbConcerts mod 2"/>
```

Lors de l'application du template d'affichage, on n'écrit l'élément que si celui-ci est dans l'interval donné.

```
<xsl:template match="article">
    <xsl:param name="AffichageBas"/>
    <xsl:param name="AffichageHaut"/>
    <xsl:if test="position()>$AffichageBas and position()&lt;=$AffichageHaut and position()&lt;=6">
        <xsl:call-template name="article"/>
    </xsl:if>
</xsl:template>
```

TRAITEMENT XSLT EN ASP.NET

Généralités

L'utilisation de XSLT dans asp.net est souvent cachée par l'utilisation d'un DataSet et du chargement du xml dans celui-ci. Cependant, il peut être intéressant de manipuler directement du XML et de procéder à une transformation XSLT plutôt que de passer par plusieurs étapes dans le cas de l'utilisation d'un DataSet.

Utilisation du composant .net

Comme tout composant .net il est possible de le mettre dans la page sous sa forme déclarative ou de code. Nous choisissons la première solution qui permet de masquer le code et rend la page plus claire

Exemple
<asp:Xml id="Destination" runat="server" TransformSource="\voyages\xsl\fiche_guide_touristique.xslt" DocumentSource= 'fiche.xml'/>

id : permet de se référencer à ce composant lors des appels
TransformSource : Source XSLT
DocumentSource : Source XML
Runat : exécution coté server

Envoyer des paramètres au composant

Il est possible d'envoyer des valeurs qui seront traitées ensuite dans le XSL

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="idarticle" select="1" />
<xsl:param name="path" select="1" />
....
```

Pour cela il faut d'abord declarer les espaces de noms avec lesquels on va travailler.

```
<%@ import Namespace="System.Xml" %>
<%@ import Namespace="System.Xml.Xsl" %>
<%@ import Namespace="System.Xml.XPath" %>
```

Ensuite dans le code lors du chargement de la page nous allons passer ces paramètres au composant

```
<script language="c#" runat="server">
void Page_Load(Object sender, EventArgs e)
{
// On récupère l'id passé en paramètre dans l'url
String FicheId = Request.QueryString["ficheid"];

   // On passe en parametre l'id de page
   XsltArgumentList xslArg = new XsltArgumentList();
   xslArg.AddParam("FicheId", "", Convert.ToInt32(FicheId));
   Destination.TransformArgumentList = xslArg;

   // On donne un source XML de manière dynamique
   Destination.DocumentSource="Fiche" +  ".xml";

}

</script>
```

TODO :

- Mise en cache
  - utilisation de plusieurs XSL sur le résultat d'une première transformation

ANNEXES

Parsers

Leçon de codage des caractères

Les formats de texte standard sont créés avec des jeux de caractères standard. N'oubliez pas que tous les ordinateurs enregistrent le texte sous forme de nombres. Toutefois, des systèmes différents peuvent enregistrer le même texte sous des formes différentes. Le tableau suivant indique comment est enregistrée une plage d'octets, d'abord sur un ordinateur standard exécutant Microsoft Windows®, à l'aide de la page de code par défaut 1252, puis sur un ordinateur standard Apple® Macintosh®, à l'aide de la page de code Macintosh Roman.

Octet	Windows	Macintosh
140	Œ	å
229	å	Â
231	ç	Á
232	è	Ë
233	é	È

Par exemple, lorsque votre grand-mère commande un livre sur le site http://www.barnesandnoble.com/, elle ne se rend pas compte que son ordinateur Macintosh n'enregistre pas ses caractères de la même manière que le nouveau serveur Web Windows 2000 sur lequel s'exécute www.barnesandnoble.com. Lorsqu'elle saisit son adresse en Suède dans le champ Adresse d'expédition du formulaire de commande Internet, elle pense qu'Internet transmettra correctement le caractère å (valeur d'octet de 140 sur son Macintosh), alors qu'en fait son message sera reçu et traité par des ordinateurs qui traduisent la valeur d'octet 140 par la lettre Œ.

Unicode

Le Consortium Unicode a décidé qu'il serait judicieux de définir une page de code universelle (qui utiliserait 2 octets par caractère, au lieu d'un seul) pour couvrir toutes les langues du monde, ce qui éliminerait définitivement ce problème de mappage entre les différentes pages de code.

Donc, si Unicode résout les problèmes du codage des caractères multiplate-forme, pourquoi n'est-ce pas devenu la norme officielle ? Premièrement, parce que le passage à Unicode peut doubler la taille des fichiers, ce qui en termes de réseau n'est pas idéal. C'est pourquoi certaines personnes préfèrent encore utiliser les anciens jeux de caractères sur un octet (par exemple, ISO-8859-1 à ISO-8859-15, Shift-JIS, EUC-KR, etc.).

La deuxième raison est qu'il existe encore beaucoup de systèmes qui ne sont pas du tout compatibles Unicode, ce qui signifie que sur un réseau, certaines valeurs d'octet qui composent les caractères Unicode peuvent causer d'énormes problèmes sur ces

anciens systèmes. C'est la raison pour laquelle l'alphabet universel UTF (Unicode Transformation Formats) a été défini ; il utilise des techniques de conversion de bits pour coder les caractères Unicode sous forme de valeurs d'octet pour les rendre « transparentes » (ou les transmettre de manière fiable) sur ces anciens systèmes.

Le plus populaire de ces systèmes de codage de caractères est UTF-8. UTF-8 prend les 127 premiers caractères de la norme Unicode (qui correspond, en fait, aux caractères latins de base, à savoir A-Z, a-z et 0-9, et à quelques caractères de ponctuation) pour les mapper directement sur des valeurs d'octet uniques. Il applique ensuite une technique de conversion de bits qui consiste à utiliser le bit fort de l'octet pour coder le reste des caractères Unicode. Il résulte de tout ceci que le petit caractère suédois å (0xE5) devient le charabia suivant, sur 2 octets Ã¥ (0xC3 0xA5). À moins d'être capable d'effectuer une conversion de tête des bits, les données codées en UTF-8 ne sont pas lisibles par l'homme.

En-tête Content-Type

Étant donné que les jeux de caractères sur un octet sont toujours utilisés, le problème de transfert des données ne sera pas résolu tant qu'il ne sera pas possible de spécifier dans quel jeu de caractères se trouvent les données. Conscients de ce problème, les groupes de protocoles HTTP et de messagerie Internet ont défini une méthode standard pour spécifier le jeu de caractères défini dans la propriété Content-Type de l'en-tête d'un message. Cette propriété spécifie un jeu de caractères de la liste des noms de jeux de caractères enregistrés, définie par IANA (Internet Assigned Numbers Authority). Ainsi, un en-tête HTTP standard peut contenir le texte suivant :

```
HTTP/1.1 200 OK

Content-Length: 15327

Content-Type: text/html; charset:ISO-8859-1;

Server: Microsoft-IIS/5.0

Content-Location: http://www.microsoft.com/Default.htm

Date: Wed, 08 Dec 1999 00:55:26 GMT

Last-Modified: Mon, 06 Dec 1999 22:56:30 GMT
```

Cet en-tête indique à l'application que ce qui vient après l'en-tête est écrit dans le jeu de caractères ISO-8859-1.

Métabalises Content-Type

La propriété Content-Type est facultative et, dans certaines applications, les informations d'en-tête HTTP disparaissent et seul le code HTML est transmis. Pour pallier ce problème, le groupe de normes HTML a défini une métabalise facultative pour permettre de spécifier le jeu de caractères dans le document HTML lui-même, en rendant le jeu de caractères du document HTML suffisamment explicite.

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
```

Dans ce cas, le jeu de caractères ISO-8859-1 déclare que dans cette page HTML particulière, la valeur d'octet 229 signifie å. Cette page est ainsi dépourvue de toute ambiguïté sur tous les systèmes et les données ne risquent pas d'être mal interprétées. Malheureusement, en raison de son caractère facultatif, cette métabalise est source d'erreurs.

Entités de caractères

Tous les systèmes ne prennent pas en charge tous les jeux de caractères enregistrés. Je ne pense pas, par exemple, qu'il existe beaucoup de plates-formes qui prennent en charge le jeu de caractères utilisé sur les gros systèmes IBM, appelé EBCDIC.

Windows NT le fait, mais il ne doit pas y en avoir beaucoup d'autres, ce qui explique pourquoi la page d'accueil http://www.fr.ibm.com/ génère du code ASCII.

Comme plan de secours, HTML permet d'encoder chaque caractère sur une page en spécifiant leur valeur Unicode exacte. Ces entités de caractères sont ensuite analysées indépendamment du jeu de caractères et leur valeur Unicode peut être déterminée sans ambiguïté. La syntaxe est « å » ou « å ».

XSLT

Eléments

xsl:import Import d'un programme XSLT
xsl:include Inclusion d'un programme XSLT
xsl:output Indique le format de sortie
xsl:param Définit un paramètre
xsl:template Définit une règle XSLT
xsl:variable Définit une variable XSLT

Exercices

Rédiger une DTD pour une bibliographie.

Cette bibliographie :

- contient des livres et des articles ;
- les informations nécessaires pour un livre sont :
  - son titre général ;
  - les noms des auteurs ;
  - ses tomes et pour chaque tome, leur nombre de pages ;
  - des informations générales sur son édition comme par exemple le nom de l'éditeur, le lieu d'édition, le lieu d'impression, son numéro ISBN ;
- les informations nécessaires pour un article sont :
  - son titre ;
  - les noms des auteurs ;
  - ses références de publication : nom du journal, numéro des pages, année de publication et numéro du journal
- on réservera aussi un champ optionnel pour un avis personnel.

Correction

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT biblio (livre | article)*>
<!ELEMENT livre (titre, auteur+, tome*, edition, avis?)>
```

```
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT tome (nb_pages)>
<!ELEMENT nb_pages (#PCDATA)>
<!ELEMENT edition (editeur, lieu_edition, lieu_impression, isbn)>
<!ELEMENT editeur (#PCDATA)>
<!ELEMENT lieu_edition (#PCDATA)>
<!ELEMENT lieu_impression (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT avis (#PCDATA)>
<!ELEMENT article (titre, auteur+, journal)>
<!ELEMENT journal (nom_journal, page, num_journal, annee)>
<!ELEMENT nom_journal (#PCDATA)>
<!ELEMENT page (#PCDATA)>
<!ELEMENT num_journal (#PCDATA)>

<!ELEMENT annee (#PCDATA)>
```

Ajout d'attributs à une DTD

1. Rédiger un document XML de quelques mots, comportant une DTD externe déclarant deux éléments :
    1. elt.racine peut contenir autant de fois que l'on veut du texte ou elt./enfant ;
    2. elt.enfant peut contenir du texte.
2. Déclarer une entité paramétrique entite1 permettant d'insérer dans la DTD la chaîne de caractères "#PCDATA|elt.enfant". Utiliser cette entité dans la déclaration de l'élément elt.racine.
3. Déclarer une entité paramétrique entite2 permettant d'insérer "<elt.enfant>entité<elt.enfant> et l'appeler dans un corps de texte d'elt.racine.
4. Déclarer l'entité de caractère Eacute comme étant le caractère É (qui correspond à É). L'appeler dans un corps de texte.

Correction

Xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE elt.racine SYSTEM "exo6.dtd">
<elt.racine>
A l'intérieur d'un élément enfant: &entite2;
Avec une entité de caractère: &entite3;
</elt.racine>
```

Dtd

```
<!ENTITY % entite1 "#PCDATA|elt.enfant">
<!ENTITY entite2 "<elt.enfant>sgrogneugneu</elt.enfant>">
<!ENTITY entite3 "&#201;">
<!ELEMENT elt.racine (%entite1;)*>
```

```
<!ELEMENT elt.enfant (#PCDATA)>
```

Ecriture d'une feuille XSL simple

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<recette>
        <entete>
                <auteur>Casimir</auteur>
                <titre>Recette du Gloubi-Boulga</titre>
                <remarque>Pour une personne</remarque>
        </entete>
        <procedure>
                Remplir un saladier avec de la confiture de fraises, du chocolat râpé,
                des bananes écrasées, de la moutarde forte, des saucisses de Toulouse
                écrasées tièdes mais crues. Mélanger vigoureusement jusqu'à obtenir une
                bouillie marron-clair. Il est normal qu'il y ait des grumeaux. Les
                proportions sont environ égales pour tous les ingrédients, mais il est
                possible de varier selon les goûts de chacun.
        </procedure>
</recette>
```

Créer une feuille de style XSL permettant à partir de cette fiche recette de produire une page HTML qui :

a pour titre le contenu de la balise titre ;
commence par un titre <h1> ayant comme contenu le contenu de la balise titre ;
donne ensuite le nom de l'auteur de la recette ;
affiche ensuite le mot Remarque : puis le contenu de la balise remarque ;
affiche Procédure en niveau <h2> ;
dans un paragraphe, présente la procédure à suivre.

Correction

```
<xsl:stylesheet version="1.0" xmlns:xsl="uri:xsl">
    <xsl:template match="/">
        <html>
            <head>
                <title>
                    <xsl:value-of select="//titre"/>
                </title>
            </head>
            <body>
                <h1>
                    <xsl:value-of select="//titre"/>
                </h1>
                <p>
                    <b>Auteur:</b>
                    <xsl:value-of select="//auteur"/>
                </p>
```

```
                <p>
                    <b>Remarque:</b>
                    <xsl:value-of select="//remarque"/>
                </p>
                <h2>Procédure</h2>
                <p>
                    <xsl:value-of select="//procedure"/>
                </p>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

XPATH & XSLT

En utilisant le fichier XML suivant réaliser chaque exercice demandé

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<systeme_solaire>
    <etoile>
        <nom>Soleil</nom>
        <type_spectral>G2</type_spectral>
        <age unit="milliard d'annees">5</age>
    </etoile>
    <planete type="tellurique">
        <nom>Mercure</nom>
        <distance unit="UA">0.4</distance>
        <masse unit="masse terrestre">0.06</masse>
        <diametre unit="diamètre terrestre">0.4</diametre>
    </planete>
    <planete type="tellurique">
        <nom>Vénus</nom>
        <distance unit="UA">0.7</distance>
        <masse unit="masse terrestre">0.8</masse>
        <diametre unit="diamètre terrestre">0.9</diametre>
    </planete>
    <planete type="tellurique">
        <nom>Terre</nom>
        <distance unit="km">149600000</distance>
        <masse unit="kg">5.98e24</masse>
        <diametre unit="km">12756</diametre>
        <satellite>1</satellite>
    </planete>
    <planete type="tellurique">
        <nom>Mars</nom>
        <distance unit="UA">1.5</distance>
        <masse unit="masse terrestre">0.1</masse>
        <diametre unit="diamètre terrestre">0.5</diametre>
        <satellite>2</satellite>
    </planete>
    <planete type="gazeuse" anneau="yes">
        <nom>Jupiter</nom>
        <distance unit="UA">5.2</distance>
        <masse unit="masse terrestre">318</masse>
        <diametre unit="diamètre terrestre">11</diametre>
        <satellite>16</satellite>
    </planete>
    <planete type="gazeuse" anneau="yes">
        <nom>Saturne</nom>
        <distance unit="UA">9.6</distance>
        <masse unit="masse terrestre">95</masse>
        <diametre unit="diamètre terrestre">9.4</diametre>
```

```
        <satellite>18</satellite>
    </planete>
    <planete type="gazeuse" anneau="yes">
        <nom>Uranus</nom>
        <distance unit="UA">19.2</distance>
        <masse unit="masse terrestre">14.5</masse>
        <diametre unit="diamètre terrestre">4</diametre>
        <satellite>15</satellite>
    </planete>
    <planete type="gazeuse" anneau="yes">
        <nom>Neptune</nom>
        <distance unit="UA">30.1</distance>
        <masse unit="masse terrestre">17.2</masse>
        <diametre unit="diamètre terrestre">3.8</diametre>
        <satellite>8</satellite>
    </planete>
    <planete type="Kuiper">
        <nom>Pluton</nom>
        <distance unit="UA">39.4</distance>
        <masse unit="masse terrestre">0.002</masse>
        <diametre unit="diamètre terrestre">0.2</diametre>
        <satellite>1</satellite>
    </planete>
</systeme_solaire>
```

Boucle

- l'aide d'une boucle <xsl:for-each>, présenter les données sous la forme d'une liste donnant pour chaque planète son nom, sa distance par rapport au Soleil, sa masse et son diamètre, de manière à obtenir le résultat suivant ;

Mercure :

- Distance au soleil: 0.4
- Masse: 0.06
- Diamètre: 0.4

Vénus :

- Distance au soleil: 0.7
- Masse: 0.8
- Diamètre: 0.9

Terre :

- Distance au soleil: 149600000
- Masse: 5.98e24
- Diamètre: 12756

Mars :

- Distance au soleil: 1.5
- Masse: 0.1
- Diamètre: 0.5

Jupiter :

- Distance au soleil: 5.2
- Masse: 318
- Diamètre: 11

Saturne :

- Distance au soleil: 9.6
- Masse: 95
- Diamètre: 9.4

Uranus :

- Distance au soleil: 19.2
- Masse: 14.5
- Diamètre: 4

Neptune :

- Distance au soleil: 30.1
- Masse: 17.2
- Diamètre: 3.8

Pluton :

- Distance au soleil: 39.4
- Masse: 0.002
- Diamètre: 0.2

Correction

```
<?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="html" version="html 4.01" encoding="ISO-8859-1" doctype-public="-//W3C//DTD HTML 4.01//EN"/>

<xsl:template match="/">
    <html lang="fr">
        <head>
            <title>Exercice 9</title>
        </head>
        <body>
        <xsl:for-each select="systeme_solaire/planete">
            <p><b><xsl:value-of select="nom"/> :</b></p>
            <ul>
```

```
                <li>Distance au soleil: <xsl:value-of select="distance"/> <xsl:value-of select="distance/@unit"/></li>
                <li>Masse: <xsl:value-of select="masse"/> <xsl:value-of select="masse/@unit"/></li>
                <li>Diamètre: <xsl:value-of select="diametre"/> <xsl:value-of select="diametre/@unit"/></li>
                <xsl:if test="satellite"><li>Nombre de satellites: <xsl:value-of select="satellite"/></li></xsl:if>
            </ul>
        </xsl:for-each>
        </body>
    </html>
</xsl:template>

</xsl:stylesheet>
```

- Ajouter à la distance, la masse et le diamètre les unités employées en récupérant la valeur de l'attribut unit pour chacun de ces éléments comme sur l'exemple suivant ;

Mercure :

- Distance au soleil: 0.4UA
- Masse: 0.06masse terrestre
- Diamètre: 0.4diamètre terrestre

Vénus :

- Distance au soleil: 0.7UA
- Masse: 0.8masse terrestre
- Diamètre: 0.9diamètre terrestre

Terre :

- Distance au soleil: 149600000km
- Masse: 5.98e24kg
- Diamètre: 12756km

Mars :

- Distance au soleil: 1.5UA
- Masse: 0.1masse terrestre
- Diamètre: 0.5diamètre terrestre

Jupiter :

- Distance au soleil: 5.2UA
- Masse: 318masse terrestre
- Diamètre: 11diamètre terrestre

Saturne :

- Distance au soleil: 9.6UA

- Masse: 95masse terrestre
- Diamètre: 9.4diamètre terrestre

Uranus :

- Distance au soleil: 19.2UA
- Masse: 14.5masse terrestre
- Diamètre: 4diamètre terrestre

Neptune :

- Distance au soleil: 30.1UA
- Masse: 17.2masse terrestre
- Diamètre: 3.8diamètre terrestre

Pluton :

- Distance au soleil: 39.4UA
- Masse: 0.002masse terrestre
- Diamètre: 0.2diamètre terrestre

Correction

```
<?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="html" version="html 4.01" encoding="ISO-8859-1" doctype-public="-//W3C//DTD HTML 4.01//EN"/>
    <xsl:template match="/">
        <html lang="fr">
            <head>
                <title>Exercice 9</title>
            </head>
            <body>
                <xsl:for-each select="systeme_solaire/planete">
                    <p>
                        <b>
                            <xsl:value-of select="nom"/> :</b>
                    </p>
                    <ul>
                        <li>Distance au soleil: <xsl:value-of select="distance"/>
                            <xsl:value-of select="distance/@unit"/>
                        </li>
                        <li>Masse: <xsl:value-of select="masse"/>
                            <xsl:value-of select="masse/@unit"/>
                        </li>
                        <li>Diamètre: <xsl:value-of select="diametre"/>
                            <xsl:value-of select="diametre/@unit"/>
                        </li>
                        <xsl:if test="satellite">
                            <li>Nombre de satellites: <xsl:value-of select="satellite"/>
                            </li>
                        </xsl:if>
                    </ul>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>
```

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

- Classer les planètes sauf la Terre par ordre croissant de leur masse ;

Pluton :

- Distance au soleil: 39.4UA
- Masse: 0.002masse terrestre
- Diamètre: 0.2diamètre terrestre
- Nombre de satellites: 1

Mercure :

- Distance au soleil: 0.4UA
- Masse: 0.06masse terrestre
- Diamètre: 0.4diamètre terrestre

Mars :

- Distance au soleil: 1.5UA
- Masse: 0.1masse terrestre
- Diamètre: 0.5diamètre terrestre
- Nombre de satellites: 2

Vénus :

- Distance au soleil: 0.7UA
- Masse: 0.8masse terrestre
- Diamètre: 0.9diamètre terrestre

Uranus :

- Distance au soleil: 19.2UA
- Masse: 14.5masse terrestre
- Diamètre: 4diamètre terrestre
- Nombre de satellites: 15

Neptune :

- Distance au soleil: 30.1UA
- Masse: 17.2masse terrestre
- Diamètre: 3.8diamètre terrestre
- Nombre de satellites: 8

Saturne :

- Distance au soleil: 9.6UA
- Masse: 95masse terrestre
- Diamètre: 9.4diamètre terrestre
- Nombre de satellites: 18

Jupiter :

- Distance au soleil: 5.2UA
- Masse: 318masse terrestre
- Diamètre: 11diamètre terrestre
- Nombre de satellites: 16

Correction

```
<?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="html" version="html 4.01" encoding="ISO-8859-1" doctype-public="-//W3C//DTD HTML 4.01//EN"/>
    <xsl:template match="/">
        <html lang="fr">
            <head>
                <title>Exercice 9</title>
            </head>
            <body>
                <xsl:for-each select="systeme_solaire/planete">
                    <xsl:sort select="masse" data-type="number"/>
                    <xsl:if test="not(nom='Terre')">
                        <p>
                            <b>
                                <xsl:value-of select="nom"/> :</b>
                        </p>
                        <ul>
                            <li>Distance au soleil: <xsl:value-of select="distance"/>
                                <xsl:value-of select="distance/@unit"/>
                            </li>
                            <li>Masse: <xsl:value-of select="masse"/>
                                <xsl:value-of select="masse/@unit"/>
                            </li>
                            <li>Diamètre: <xsl:value-of select="diametre"/>
                                <xsl:value-of select="diametre/@unit"/>
                            </li>
                            <xsl:if test="satellite">
                                <li>Nombre de satellites: <xsl:value-of select="satellite"/>
                                </li>
                            </xsl:if>
                        </ul>
                    </xsl:if>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```