

Introduction à XML [DVIR3]

Informatique et systèmes
finalité Réseaux &
télécommunication

2016-2017

XML

- 1.Introduction
- 2.Langage XML
- 3.Document XML
- 4.DTD : Déclaration de la structure du document
- 5.XML Schema
- 6.Adressage de fragments xml : Xpath
- 7.Contenu et présentation : CSS et XSL
- 8.Langage de transformation XSLT
- 9.Langage de formatage XSL-FO
- 10.PHP - XML

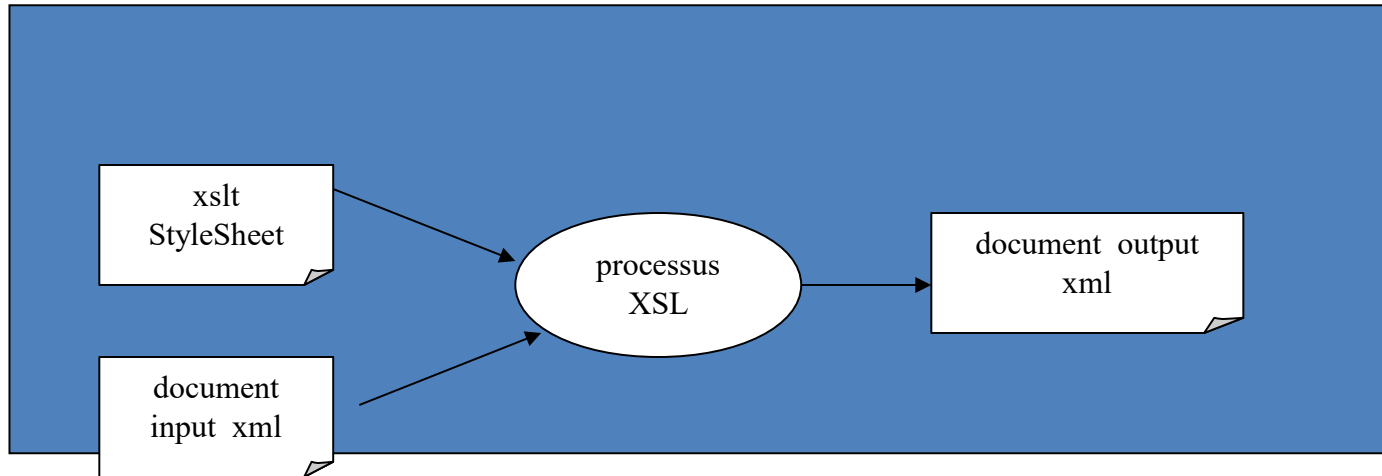
XML

- 1.Introduction
- 2.Langage XML
- 3.Document xml
- 4.DTD : Déclaration de la structure du document
- 5.XML Schema
- 6.Adressage de fragments xml : Xpath
- 7.Contenu et présentation : CSS et XSL**
- 8.Langage de transformation XSLT
- 9.Langage de formatage XSL-FO
- 10.PHP - XML

Contenu et présentation : CSS et XSL

2

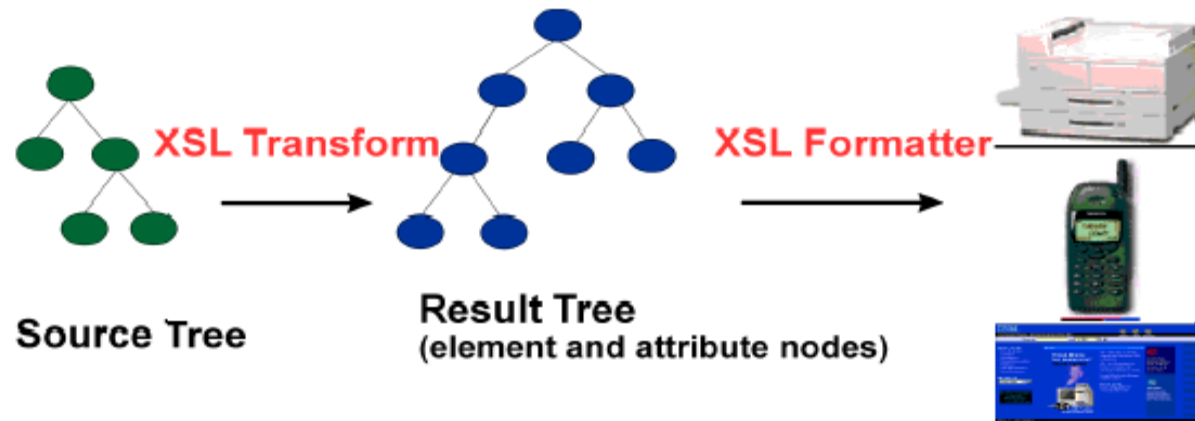
- XSL= eXtensible Stylesheet Language



- Norme de référence : <http://www.w3.org/TR/xsl>

2 processus XSL

3



- Langage de construction de feuille de style
- Une feuille de style
 - définit la présentation d'une classe de documents et
 - décrit comment une instance de cette classe doit être transformée pour arriver à cette présentation.

Parties de XSL

5

- Un langage de transformation d'un document XML en un autre document d'autre structure
- Un vocabulaire pour décrire une sémantique de formatage.

Étapes d'un processus XSL

6

- **1. étape de transformation :**

le langage de transformation est utilisé pour transformer le document XML en un autre document basé sur le vocabulaire défini par la norme XSL

- **2. étape de formatage :**

du document intermédiaire en document final selon la présentation souhaitée

Feuille de style

7

=

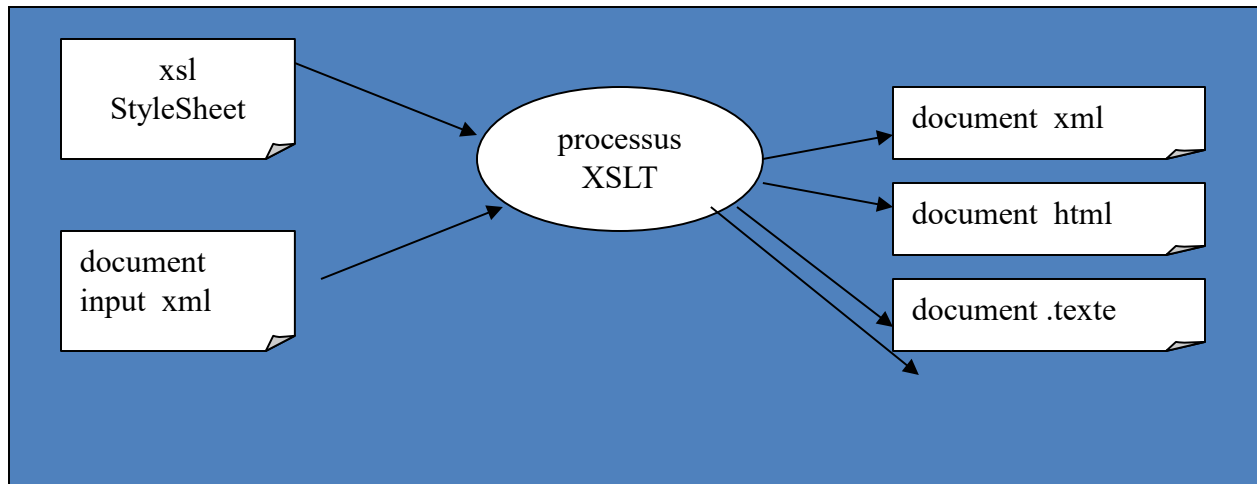
document xml respectant une structure particulière définie dans la norme xsl

- *xsl:élt* → élt(s) préfixés de ce *namespace* ∈ au langage de transformation
- *fo:élt* → élt(s) préfixés de ce *namespace* ∈ au vocabulaire de formatage

Langage de transformation

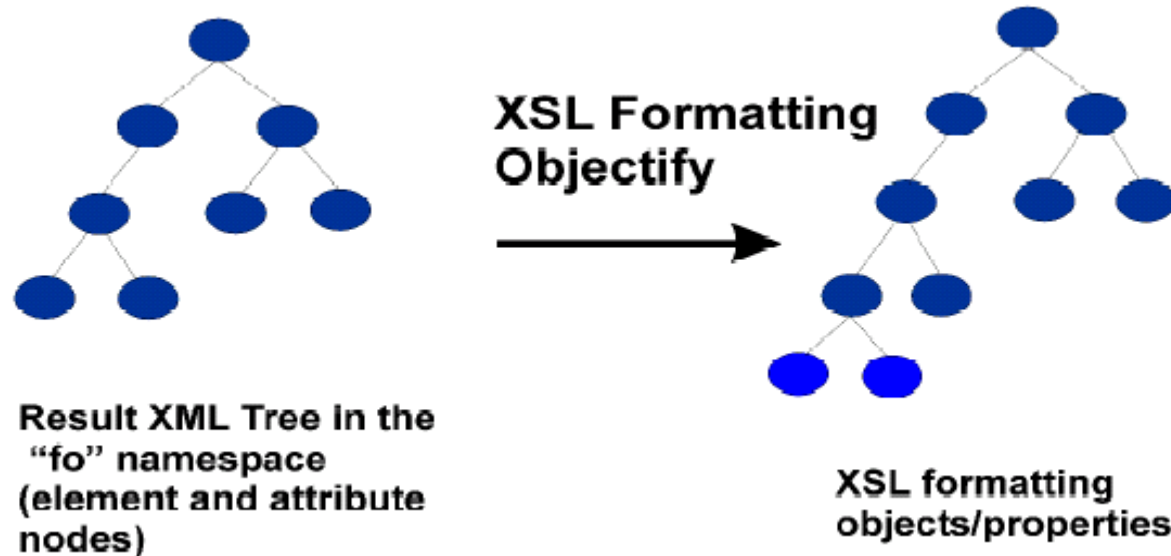
8

- Normalisé par la recommandation XSLT
<http://www.w3.org/TR/xslt>



Langage de formatage

9



- les éltls (objets de formatage) définissent des pages, des §, des tables,....
- les propriétés de formatage définissent la taille, les marges, la police, la couleur, le style de caractère,...

XML

10

- 1.Introduction
- 2.Langage XML
- 3.Document xml
- 4.DTD : Déclaration de la structure du document
- 5.XML Schema
- 6.Adressage de fragments xml : Xpath
- 7.Contenu et présentation : CSS et XSL
- 8.Langage de transformation XSLT**
- 9.Langage de formatage XSL-FO
- 10.PHP - XML

- Document :

<http://www.w3.org/TR/XSLT>

<http://xmlfr.org/w3c/TR/xslt/>

Principe

12

- XSLT est un langage complet qui définit la syntaxe et la sémantique d'une feuille de style utilisée pour transformer un document XML en un autre document XML,(X)HTML, texte,...
- XSLT est utilisé au sein du langage XSL qui définit aussi un vocabulaire pour décrire des éléments de formatage

Transformation

13

- Effectuée dans le navigateur client qui affiche le fichier xml en utilisant une feuille de style XSLT associée (MSIE v.5.5)
- Effectuée sur le serveur web qui produit dynamiquement du html envoyé au navigateur client
- Effectuée dans un parseur indépendant sur serveur, qui produit du html placé ensuite sur le serveur Web (xp de James Clarck).

Appel d'une feuille de style

14

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

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

...

Structure d'une feuille de style

15

- Comportant une phase de transformation
Élément racine : `xsl:stylesheet` *ou* `xsl:transform`

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

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

...

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

Structure d'une feuille de style complète

16

- Comportant aussi une phase de formatage

```
<?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:stylesheet>
```

Éléments de plus haut niveau

17

- Élts qui permettent d'importer et d'inclure d'autres feuilles de style
<xsl:import> et *<xsl:include>*
- Élts déclaratifs :
<xsl:strip-space>, *<xsl:preserve-space>*
<xsl:output>, *<xsl:key>*, *<xsl:decimal-format>*,
<xsl:namespace-alias>, *<xsl:attribute-set>*,
<xsl:param>, *<xsl:variable>*
- Règles de templates (templates rules)
<xsl:template>

Templates rules

18



Structure d'une règle de *template*

19

```
<xsl:template match="pattern">                ← instruction
  ...
  template
  ...
</xsl:template>
```

Pattern

contient une expression qui permet de savoir si le modèle ou template s'applique à certains nœuds de l'arbre source (cf. Xpath)

Template

contient des élt (html,..) et des données textuelles (carnet d'adresse,...) à recopier dans l'arbre résultat

Exemple XSLT

2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">                                <!-- noeud contexte →
    <html>
      <head><title>Exemple de feuille de style</title></head>
      <body>
        <h1> Carnet d'adresses </h1>
        <xsl:for-each select="carnet/personne"> <!-- répétition pour chaque nœud →
          <p><xsl:value-of select="nom"/> <!-- création d'un nœud texte →
            <xsl:text>,</xsl:text>
            <xsl:value-of select="prenom"/><br/>
            <xsl:value-of select="adresse/num"/> <xsl:text> </xsl:text>
            <xsl:value-of select="adresse/rue"/> <xsl:text>,</xsl:text>
            <xsl:value-of select="adresse/codepostal"/> <xsl:text> </xsl:text>
            <xsl:value-of select="adresse/ville"/></p>
          </xsl:for-each>
        </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

Principe de fonctionnement d'un modèle ou *template*

21

- Le parseur XSLT démarre le processus en appliquant le *template* correspondant au *pattern* du nœud racine "/"
- Les éléments non préfixés par "xsl:" , les données textuelles rencontrées et les nœuds générés par les instructions "xsl:" sont insérés dans l'arbre résultat
- Ces instructions peuvent sélectionner d'autres nœuds de l'arbre source et leur appliquer d'autres *templates*. etc...
<xsl:apply-templates>

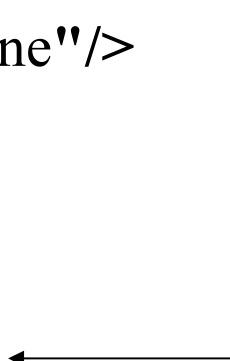
Fonctionnement d'un <xsl:apply-templates>

22

Sélection d'un ensemble de nœuds à l'aide d'un chemin d'accès Xpath

```
<xsl:template match="/">
  <html>
    <body>
      <h1>liste des noms</h1>
      <xsl:apply-templates select="/carnet/personne"/>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="personne">
  <xsl:value-of select="nom"/>,<xsl:value-of select="prenom"/><br/>
</xsl:template>
```



Les patterns

23

- ou *motifs* sont des expressions Xpath de type chemin d'accès qui permettent de sélectionner et d'appliquer un *template* particulier.
- n'utilisent que les axes *child::* *attributes::* et le raccourci //

Sélection du *template*

- Les *templates* candidats les plus appropriés lors du traitement d'un nœud sont recherchés dans toute la feuille de styles et sélectionnés grâce aux *patterns*
- Un *pattern* correspond à un nœud s'il existe un "contexte possible" dans lequel le calcul de l'expression donnée par le pattern fournit comme résultat le nœud testé ou un de ses ancêtres

Contexte possible

25

L'application via `<xsl:apply-templates select="/carnet/personne"/>`
au 1er nœud "personne" d'un des *templates* suivants :

```
<xsl:template match="personne">... </xsl:template>
```

```
<xsl:template match="*">... </xsl:template>
```

```
<xsl:template match="carnet/personne">... </xsl:template>
```

```
<xsl:template match="carnet/personne[position()=1]">... </xsl:template>
```

```
<xsl:template match="/carnet/personne">... </xsl:template>
```

est possible car il existe un "contexte possible" constitué de "carnet" ou
"/" (ancêtre de personne) qui est tel que l'expression du pattern contienne
ce 1er nœud

Contexte impossible

La sélection via `<xsl:apply-templates select="/carnet/personne"/>`
au 1er nœud "personne" d'un des *templates* suivants :

```
<xsl:template match="erreur/personne">... </xsl:template>
```

```
<xsl:template match="carnet/personne[position()=2]">... </xsl:template>
```

```
<xsl:template match="/carnet/personne/nom">... </xsl:template>
```

est impossible car il n'existe pas de "contexte possible" qui est tel que l'expression du pattern contienne ce 1er nœud

Résolution de conflits si plusieurs *templates* sont sélectionnés

27

Règle de précedence d'importation

Un template défini par `<xsl:import>` est "éliminé" si **un template est défini dans la feuille de style avec le même pattern**

Priorité explicite (*priority*)

```
<xsl:template pattern="personne" priority="1">... </xsl:template>  
<xsl:template pattern="personne" priority="2">... </xsl:template>
```

Priorité implicite par le parseur

aux patterns qui donnent le plus de détails quant à la sélection à effectuer

```
<xsl:template pattern="carnet/personne[position()=1]">... </xsl:template>  
<xsl:template pattern="carnet/personne">... </xsl:template>
```

Priorité implicite

28

- Priorité = 0 si le pattern est un nom d'*élément*, d'un *attribut* éventuellement précédés d'un *namespace* ou s'il vaut "*processing-instruction*(nom)"
- Priorité = -0.25 si le pattern est le nœud "*" avec un namespace (ex.: *fo:**)
- Priorité = -0.5 si le pattern est le nœud "*" sans namespace ou s'il est égal à "*comment*()", "*text*()", "*processing-instruction*()" ou "*node*()"
- Priorité = 0.5 dans les autres cas

Exercices

29

- A partir du carnet d'adresse *carnet.xml*, réécrire la valeur de tous les nœuds sauf pour les éléments "codepostal" qui seront précédés d'un nœud texte "B- "
- Concevoir une page *.html* qui reprend pour chaque AA organisée,
 - l'id,
 - l'intitulé,
 - le nombre d'heures,
 - le quadrimestre,
 - les enseignants associés

Résolution de conflits si plusieurs *templates* sont sélectionnés

30

Avec l'attribut *mode* : seuls les templates qui possèdent la même valeur d'attribut *mode* seront sélectionnés

```
<xsl:template match="/">
  <html>
    <body>
      <h1>liste des noms</h1>
      <xsl:apply-templates select="/carnet/personne"
                           mode="prenom_nom"/>
    </body>
  </html>
</xsl:template>
```


Attribut *mode*

31

```
<xsl:template match="personne">
  <xsl:value-of select="nom"/>
  <xsl:value-of select="prenom"/><br/>
</xsl:template>
```

```
<xsl:template match="personne" mode="prenom_nom">
  <b>
    <xsl:value-of select="prenom"/> <xsl:text> , </xsl:text>
    <xsl:value-of select="nom"/><br/>
  </b>
</xsl:template>
```

Templates par défaut

32

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

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

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

```
<xsl:template match="processing-instruction() | comment()">  
</xsl:template>
```

Exemple

33

```
<liste-messages-panne>
  <message>
    <description>FUI TE HUI LE MOTEUR</description>
    <marque-vehicule>PEUGEOT</marque-vehicule>
    <type>MAU000AKL683</type>
    <numero-panne>HM003</numero-panne>
    <procedure-reparation href="pr.xml">PSA HM003-01</procedure-reparation>
  </message>
  <message>
    ...
  </message>
</liste-messages-panne>
```

Instructions des templates

34

- Définitions de variables et de paramètres
 - <xsl:variable>
 - <xsl:param>
- Instructions relatives aux templates
 - <xsl:apply-templates>
 - <xsl: call-template >
 - <xsl:with-param>
 - <xsl:sort>
- Instructions répétitives
 - <xsl:for-each>

Instructions des templates

35

- Instructions conditionnelles
 - `<xsl:if>`
 - `<xsl:choose>`
 - `<xsl:when>`
 - `<xsl:otherwise>`
- Instructions créatrices d'éléments dans l'arbre résultat
 - `<élément>` *non préfixé*
 - `<xsl:element>`
 - `<xsl:attribute>`
 - `<xsl:copy>`
 - `<xsl:copy-of>`
- Instructions créatrices de nœuds textes dans l'arbre résultat
 - nœuds textes dans un template
 - `<xsl:text>`
 - `<xsl:value-of>`
 - `<xsl:number>`

Instructions des templates

36

- Instructions créatrices de nœuds autres dans l'arbre résultat
 - <xsl:processing-instruction>*
 - <xsl:comment>*
- Instructions qui gèrent des clés de références croisées
 - <xsl:key>*
- Instructions propres aux templates importés
 - <xsl:apply-imports>*
- Instructions autres
 - <xsl:message>*
 - <xsl:fallback>*

Arbre résultat

37

- Est produit par une transformation XSLT
- Possède une structure et un format déterminé par l'élément `<xsl:output method="html|xml|text">`
- L'instruction `<?xml.....?>` est créée automatiquement.
- Le format des éléments générés est adapté au type sélectionné
ex. : é -> é *en html*

Élément racine

38

<xsl:stylesheet> ou <xsl:transform>

<xsl:stylesheet version="version"

[id="id"

extension-element-prefixes="préfixes"

exclude-result-prefixes="préfixes"] >

[xsl:import ,

top-level-element]

</xsl:stylesheet>

Élément racine :exemple

39

```
<xsl:stylesheet version="1.0"  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
  xmlns:xt="http://www.jclark.com/xt"  
  extension-element-prefixes="xt"  
  exclude-result-prefixes="prefixe">
```

```
<xsl:import href="http://www.xyz.be/xls/sheet2.xml">
```

```
  top-level-elements
```

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

Intégration de .xsl dans .xml

40

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<?xml-stylesheet type="text/xml" href="#feuilleDeStyle"?>
```

```
<carnet>
```

```
...
```

```
<xsl:stylesheet version="1.0" id="feuilleDeStyle"
```

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

```
...
```

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

```
...
```

```
</carnet>
```

Formatage sélectif : modes de traitement (1/3)

41

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output="html" encoding="UTF-8"/>
<xsl:template match="/PLANETES">
  <HTML>
    <HEAD><TITLE>Le tableau des planètes</TITLE></HEAD>
    <BODY>
      <H1>
        Le tableau des planètes
      </H1>
      <TABLE BORDER="2">
        <TR>
          <TD>Nom</TD>
          <TD>Masse</TD>
        </TR>
        <xsl:apply-templates/>
      </TABLE>
    </BODY>
  </HTML>
</xsl:template>
```

Modes de traitement (2/3)

42

```
<xsl:template match="PLANETE">
  <xsl:if test="NOM='Terre'">
    <TR>
      <TD><xsl:apply-templates select="NOM" mode="gras" /></TD>
      <TD><xsl:apply-templates select="MASSE" mode="gras" /></TD>
    </TR>
  </xsl:if>
  <xsl:if test="NOM!='Terre'">
    <TR>
      <TD><xsl:apply-templates select="NOM" /></TD>
      <TD><xsl:apply-templates select="MASSE" /></TD>
    </TR>
  </xsl:if>
</xsl:template>
```

Modes de traitement (3/3)

43

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

```
<xsl:template match="MASSE">  
  <xsl:value-of select="."/>  
  <xsl:text> </xsl:text>  
  <xsl:value-of select="@UNITE"/>  
</xsl:template>
```

```
<xsl:template match="NOM" mode="gras">  
  <b><xsl:value-of select="."/></b>  
</xsl:template>
```

```
<xsl:template match="MASSE" mode="gras">  
  <b> <xsl:value-of select="."/><xsl:text> </xsl:text>  
    <xsl:value-of select="@UNITE"/></b>  
</xsl:template>
```

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

xsl:variable

44

```
<xsl:variable name="nom" select="expression XPath">
```

ou template à exécuter si l'attribut select est absent

```
</xsl:variable>
```

<xsl:variable> avec "select"

45

- `<xsl:variable name="addr" select="adresse/*"/>`
- `<xsl:variable name="moyenne" select="($x+$y)/2"/>`

<xsl:variable> avec "template"

46

```
<xsl:variable name="name">
  <i><xsl:value-of select="nom"/></i>,
  <xsl:value-of select="prenom"/> <br/>
</xsl:variable>
```

- Le fragment d'arbre généré est constitué de nœuds éléments <i> et
 ainsi que de nœuds textes.
- On lui attribue une racine fictive "/" qui contient les éléments précités

<xsl:param>

47

~ <xsl:variable>

Sauf que l'élément param

- Est
 - de plus haut niveau *ou*
 - première instruction dans un <xsl:template>
- Peut avoir une valeur par défaut

<xsl:param> dans un template

48

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates select="/carnet/personne[@titre='M.']">
      <xsl:with-param name="titre">Monsieur </xsl:with-param>
    </xsl:apply-templates>
  </xsl:template>
  <xsl:template match="personne">
    <xsl:param name="titre"><xsl:value-of select="@titre"/></xsl:param>
    <titre><xsl:value-of select="$titre"/></titre>
    <nom><xsl:value-of select="nom"/> </nom>
  </xsl:template>
</xsl:stylesheet>
```

Utilisation d'une variable

49

Soit : `<xsl:value-of select="chapter[$index]/title"/>`

- `<xsl:variable name="index">2</xsl:variable>`

‡

- `<xsl:variable name="index" select="2"/>`

La 1ère expression `$index` représente un fragment d'arbre qui, utilisé seul dans un prédicat, est converti en un booléen (true ~ 1)

Visibilité de variable ou param

50

- Visible de partout si élément de plus haut niveau
- Visible au sein du template ou des templates inclus si élément défini dans un template

Instruction <xsl:output>

51

Détermine le format du document résultat

```
<xsl:output version="version" encoding="encodage"  
  standalone="yes|no" doctype-public="public-id"  
  doctype-system="uri" media-type="mime_type"  
  indent="yes|no" cdata-section-elements="noms"  
  method="xml|html|text|nom"  
  omit-xml-declaration="yes|no" />
```

Instruction <xsl:strip-space>

52

Donne une liste de noms d'éléments de l'arbre source, séparés par un espace pour lesquels on élimine les nœuds texte contenus dans ceux-ci qui ne contiennent que des "blancs"

```
<xsl:strip-space elements="noms"/>
```

Instruction <xsl:preserve-space>

53

Donne une liste de noms d'éléments de l'arbre source, séparés par un espace, pour lesquels on conserve tous les nœuds texte contenus dans ceux-ci, même ceux qui ne contiennent que des "blancs"

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

Cas par défaut, sauf si une instruction <xsl:strip-space> est utilisée

Instruction <xsl:namespace-alias>

54

Permet de donner un alias à un préfixe de namespace si l'on désire créer une feuille de styles à partir d'une autre feuille de styles

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


Instruction <xsl:template>

55

Règles de template

```
<xsl:template match="pattern" name="nom"  
                priority="nombre"  
                mode="nom" >
```

```
    xsl:param
```

```
    template
```

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

Instruction <xsl:apply-templates>

56

Permet de sélectionner un ensemble de nœuds et de leur appliquer le(s) template(s) le(s) plus appropriés

```
<xsl:apply-templates select="expression" mode="nom">
```

```
    (xsl:sort  xsl:with-param name)*
```

```
</xsl:apply-templates>
```

Instruction `<xsl:call-template>`

57

Permet d'exécuter directement un *template* en l'appelant par un nom qui est donné par l'attribut *name* défini dans l'instruction `<xsl:template>`

```
<xsl:call-template name="nom">
```

```
  xsl:with-param name*
```

```
</xsl:call-template>
```

<xsl:call-template> : exemple

58

```
<xsl:call-template name="outAdresse">  
  <xsl:with-param name="twoLines">yes</xsl:with-param>  
</xsl:call-template>
```

```
<xsl:template name="outAdresse">  
  <xsl:param name="twoLines">non</xsl:param>  
  
  <xsl:value-of select="adresse/rue"/> <xsl:text> </xsl:text>  
  <xsl:value-of select="adresse/num"/> <xsl:text>, </xsl:text>  
  <xsl:if test="$twoLines='yes'"><br/> </xsl:if>  
  <xsl:value-of select="adresse/codePostal"/><xsl:text> </xsl:text>  
  <xsl:value-of select="adresse/localite"/><br/>  
</xsl:template>
```

Instruction <xsl:sort>

59

Trie les nœuds sélectionnés et permet de leur appliquer ensuite un template dans cet ordre

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

<xsl:sort> : exercice

60

A partir du carnet d'adresse, afficher le nom des personnes suivi du nom de leur localité de résidence en triant d'abord par nom de commune, puis par nom et prénom des personnes

Instruction répétitive <xsl:for-each>

61

<xsl:for-each select="expression">

 xsl:sort*

 template

</xsl:for-each>

Instruction conditionnelle <xsl:if>

62

```
<xsl:if test="expression">
```

```
    template
```

```
</xsl:if>
```

Exemple, exécuter le template si le nœud courant possède un attribut titre :

```
<xsl:if test="titre='@titre' ">
```

```
    <b><xsl:value-of select="@titre"/></b>
```

```
</xsl:if>
```


Instruction conditionnelle

`<xsl:choose><xsl:when><xsl:otherwise>`

63

`<xsl:choose>`

`<xsl:when test="expression">`

template

`</xsl:when>`

`<xsl:when> </xsl:when>`

`<xsl:otherwise>`

template

`</xsl:otherwise>`

`</xsl:choose>`

Instructions qui créent des éléments dans l'arbre résultat

64

- Un élément non préfixés par *xsl:* ou un élément avec un autre préfixe (non mentionné dans l'attribut *extension-element-prefixes* de l'élément `<xsl:stylesheet>`) génère un élément de même nom et de mêmes attributs dans l'arbre résultat.
- `<xsl:element>`
- `<xsl:attribut>`

Eléments non préfixés par *xsl:*

65

```
<non-xsl-element attrib1="val1" attr2="val2"...  
    use-attribute-sets="noms"  
    xsl:exclude-result-prefixes="prefixes">
```

template

```
</non-xsl-element>
```

Exemple

66

```
<liste-messages-panne>
  <message>
    <description>FUIITE HUILE MOTEUR</description>
    <marque-vehicule>PEUGEOT</ marque-vehicule >
    <type>MAU000AKL683</ type >
    <numero-panne>HM003</numero-panne>
    <procedure-reparation href="pr.xml">PSA HM003-01
    </procedure-reparation>
  </message>
  <message>
    ...
  </message>
</liste-messages-panne>
```

<xsl:element>

67

```
<xsl:element name="nom_généré" namespace="uri" use-attribute-sets="noms">  
  template  
</xsl:element>
```

Exemple :

```
<xsl:element name="blockquote">  
  <xsl:element name="b">  
    <xsl:element name="tel{position()}">022191546</xsl:element>  
  </xsl:element>  
</xsl:element>
```



```
<blockquote><b><tel3>022191546</tel3></b></blockquote>
```

<xsl:attribute>

```
<xsl:attribute name="nom_généré" namespace="uri" >  
    template générant un nœud texte*  
</xsl:attribute>
```

- Permet d'ajouter un attribut à un élément généré dans l'arbre résultat
- Permet de définir les attributs dans un ensemble d'attributs (déclarés par *<xsl:attribute-set>*)

<xsl:attribute> : exemple

69

```
<img>
  <xsl:attribute name="src">
    <xsl:text> image1</xsl:text>
    <xsl:if test="$highres='true' ">
      <xsl:text>_hq</xsl:text>
    </xsl:if>
    <xsl:text>.jpg</xsl:text>
  </xsl:attribute>
</img>
```



si highres vaut true, le résultat sera :

```
</img>
```

sinon, il sera :

```
</img>
```

<xsl:attribute-set>

70

```
<xsl:attribute-set name="nom_généré" use-attribute-sets="noms">  
  xsl:attribute*  
</xsl:attribute-set>
```

Permet de définir un ensemble d'attributs qui pourront être utilisés dans d'autres instructions

Example

71

```
<xsl:attribute-set name="normal"
  <xsl:attribute name="face">Arial</xsl:attribute>
  <xsl:attribute name="size">12</xsl:attribute>
</xsl:attribute-set>
```

```
<xsl:template match="p">
  <xsl:element name="p">
    <xsl:element name="font" use-attribute-sets="normal">
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

`<p>coucou</p>` → `<p>coucou</p>`

<xsl:copy>

72

```
<xsl:copy use-attributes-sets="noms">  
    template  
</xsl:copy>
```

- Copie le nœud courant dans l'arbre résultat, incluant les namespaces mais pas les attributs, ni les enfants (à copier explicitement dans des templates)
- *use-attributes-sets* permet d'ajouter à l'élément copié de(s) ensemble(s) d'attributs donnés par des *noms* séparés par des espaces. Ces noms doivent être définis par <xsl:attribute-set>

<xsl:copy> : exemple

73

Exécution d'un même template à un noeud courant, non explicitement

Nommé, qui est <titre1>, <titre2> ou <titre3>

```
<xsl:template match="titre1 | titre2 | titre3">
```

```
  <xsl:copy>
```

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

```
  </xsl:copy>
```

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

<xsl:copy-of>

74

<xsl:copy-of select="*expression*" />

- Copie un objet (un fragment d'arbre , un ensemble de nœuds, nombre, booléen,...) donné par un Xpath dans l'arbre résultat sous forme d'un nœud texte

- Exemple :

<xsl:copy-of select="/carnet/personne[nom='Sim']/adresse"/>

<adresse>

 <rue>rue Royale </rue>

 <numero>67</numero>

 <codepostal>1000</codepostal>

 <localite>Bruxelles</localite>

</adresse>

Instructions qui créent des nœuds textes dans l'arbre résultat

75

- Nœuds textes dans un template
- Élément `<xsl:text>`

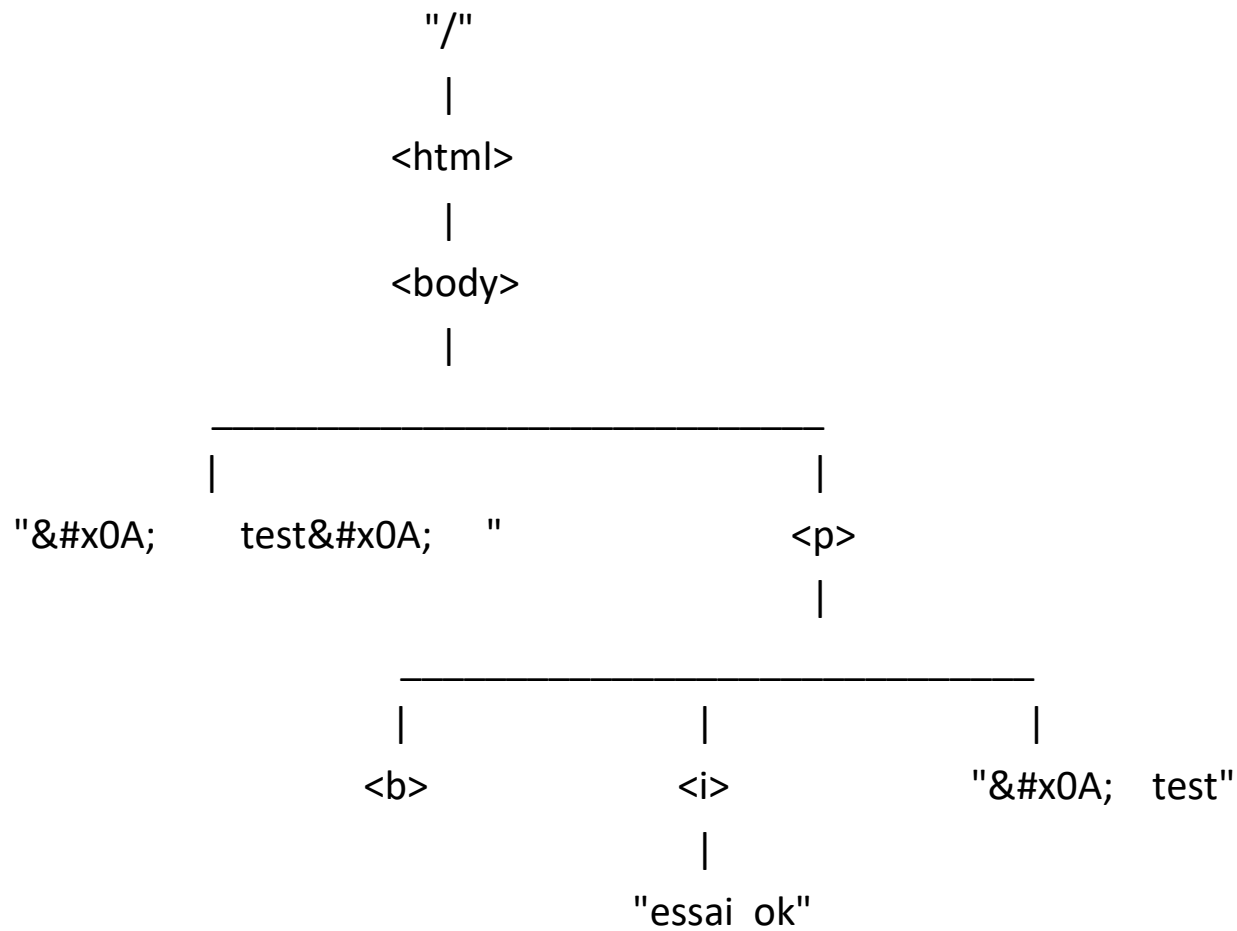
Example

76

```
<xsl:template match="/">
  <html>
    <body>
      test
      <p><b>
        </b> <i>essai<xsl:text> ok</xsl:text></i>
      test</p>
    </body>
  </html>
</xsl:template>
```

Exemple : arbre résultat

77



<xsl:text>

78

```
<xsl:text disable-output-escaping="yes|no">
```

```
  #PCDATA
```

```
</xsl:text>
```

L'instruction crée un nœud texte dans l'arbre résultat contenant les caractères définis par un ensemble de type #PCDATA

xsl:value-of

79

`<xsl:value-of select="expression" disable-output-escaping="yes|no">`

- Cette instruction permet de créer un nœud texte dans l'arbre résultat à partir d'une expression Xpath dont le résultat est converti en une chaîne de caractères
- *disable-output-escaping* est optionnel
 - "yes" : un caractère spécial (comme "<") sera gardé tel quel
 - "no" : un caractère spécial sera interprété (comme "<")

xsl:number

80

```
<xsl:number value=" "
            level="single | multiple | any"
            count="pattern" from="pattern"
            format="format" lang="lang"
            letter-value="alphabetic | traditional"
            grouping-seperator="char" grouping size="number" />
```

- Permet d'insérer un nombre formaté, à l'aide d'un nœud texte, dans l'arbre résultat
- Permet de réaliser une numérotation automatique pour des sections, chapitres,...

xsl:number : exemple

81

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

```
    Section <xsl:number/>
```

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

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

```
    Chapitre <xsl:number    level="multiple"
                           count="section|chapitre" format="1.a"/>
```

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



Section 1

Chapitre 1.a

Chapitre 1.b

Chapitre 1.c

Section 2

Chapitre 2.a

Chapitre 2.b

<xsl:number> : exercice

82

Numérotez votre carnet d'adresses trié sur le nom, prénom

1. Armillaire Lola
2. Armillaire Stella
3. Manfroid Gérard
- 4.

Instructions qui créent des instructions de traitement et commentaires dans l'arbre résultat 83

- `xsl:processing-instruction`
- `xsl:Comment`

xsl:processing-instruction

84

```
<xsl:processing-instruction name="nom">
```

```
    template
```

```
</xsl:processing-instruction>
```

Crée un nœud instruction de traitement dans l'arbre résultat

Exemple :

```
<xsl:processing-instruction name="xt">
```

```
    version="1.0"
```

```
</xsl:processing-instruction>
```



```
<?xt version="1.0"?>
```

xsl:comment

85

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

Crée un nœud commentaire dans l'arbre résultat

Exemple :

```
<xsl:comment> Généré automatiquement </xsl:comment>
```

↓

```
<!-- Généré automatiquement -->
```

```
<xsl:key name="name" match="pattern" use="expression" />
```

Instruction qui génère les clés de références croisées

- l'attribut *match* est un pattern qui détermine l'ensemble des nœuds concernés par cette clé
- L'attribut *use* est une expression Xpath qui détermine la valeur de la clé, calculée une fois pour toute, pour chacun des nœuds correspondant au pattern

xsl:key : exemple

87

<!-- génération d'une clé personne dont la valeur est calculée sur base de l'élément nom-->

```
<xsl:key name="personne" match="/carnet/personne" use="nom"/>
```

<!-- utilisation d'une clé personne pour retrouver l'élément nom-->

```
<xsl:value-of select="key('personne','Dupont')/adresse/codepostal"/>
```

```
<xsl:text> </xsl:text>
```

```
<xsl:value-of select="key('personne','Dupont')/adresse/localite"/>
```

Instructions d'inclusion et d'importation

88

```
<xsl:include href="uri "/>
```

Permet d'inclure une autre feuille de styles dans la feuille courante

```
<xsl:import href="uri "/>
```

Permet d'importer une autre feuille de styles dans la feuille courante. Ici, les templates de la feuille importée sont de priorité moindre que ceux de la feuille courante

```
<xsl:apply-imports/>
```

Permet d'exécuter un template importé qui aurait été masqué par le template courant (de priorité + élevée)

<xsl:apply-imports/> : exemple

89

```
<xsl:template match="personne">                                imported-sheet.xsl  
  <xsl:value-of select="nom"/>,  
  <xsl:value-of select="prenom"/><br/>  
</xsl:template>
```

```
<xsl:import href="imported-sheet.xsl"/>    courant-sheet.xsl  
...  
<xsl:template match="personne">  
  <hr/>  
  <b><xsl:apply-imports/> </b>  
</xsl:template>
```

xsl:message

90

```
<xsl:message terminate="yes|no">  
    template  
</xsl:message>
```

Permet d'envoyer un message (boîte de dialogue, fichier log,... fct du parseur XSLT) à l'utilisateur de la feuille de style

L'attribut *terminate* (yes) permet d'arrêter le déroulement de la feuille de style

xsl:fallback

91

```
<xsl:fallback>
```

```
    template
```

```
</xsl:fallback>
```

Permet d'exécuter le template en cas de mode "dégradé" pour des instructions non encore définies dans la version 1.0 de XSLT, par exemple

Ne fait donc rien en général

Accessibles par Xpath :

- *nodeset* **document** (*object*, *nodeset?*)
- *nodeset* **key** (*string*, *object*)
- *nodeset* **current** ()
- *string* **unparsed-entity-uri** (*string*)
- *string* **generate-id** (*nodeset?*)
- *object* **system-property** (*string*)
- *string* **format-number** (*number*, *string*, *string?*)
- *boolean* **element-available** (*string*)
- *boolean* **function-available** (*string*)

nodeset **document** (*object*, *nodeset*?)

93

Permet d'accéder à d'autres documents XML à partir de la feuille de styles

- *object* : expression Xpath.
 - si le résultat n'est pas égal à un ensemble de nœuds, il est converti en un string URI donnant l'accès à un document qui est analysé pour former un ensemble de nœuds qui sera retourné comme résultat de la fonction.
 - sinon, chacun des nœuds obtenus est converti en un string URI. La fonction retournera un document comme valeur u n ensemble de nœuds générés par l'appel successif de la fonction `document()`
- *nodeset ?* (nœud courant si absent) sert de référence pour résoudre les URI relatives

document () : exemple

94

```
<xsl:variable name="adresses" select="document('adresses.xml')"/>
```

...

```
<xsl:value-of select="$adresses/personne[nom='Dupont']/adresse/localite"/>
```


nodeset **key** (*string*, *object*)

95

Permet d'accéder à un élément à l'aide d'une clé

- *String* : précise la clé (cf. <xsl:key>)
- *Object* :
 - s'il ne représente pas un ensemble de nœuds, le résultat sera égal à l'ensemble des nœuds qui possèdent une clé dont le nom est égal au premier argument et dont la valeur est égale au 2ème argument, converti en un string
 - sinon, le résultat de l'appel est l'union des résultats produits par l'appel successif de la fct key() pour chacun de ces nœuds (2ème param=string)

nodeset **current** ()

96

- Retourne un ensemble de nœuds qui ne contient que le nœud courant
- Utile pour représenter ce nœud courant au sein des étapes d'une expression Xpath
- Fonction non utilisable dans un *pattern*

current () : exemple

97

Afficher pour chaque personne la liste des noms des autres personnes qui habitent la même commune

```
<xsl:template match="personne">
  <b><xsl:value-of select="nom"></b><hr/>
  <xsl:for-each select="../personne[adresse/codepostal=
    current()/adresse/codepostal][.!=current()]/nom"/>
    <xsl:value-of select="nom"><br/>
  </xsl:for-each>
</xsl:template>
```

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

98

Retourne l'URI d'une entité externe non analysable de nom donné
string

string **generate-id** (*nodeset?*)

99

Retourne une chaîne de caractères unique (fct du parseur XSLT) qui représente le 1er nœud de l'ensemble de nœuds donné en paramètre ou, le cas échéant, du nœud courant

object system-property (string)

100

Retourne la valeur d'une propriété système, liée au parseur XSLT, fonction de l'argument *string* :

- `xsl:version` : retourne la version du parseur XSLT (1.0)
- `xsl:vendor` : retourne l'identificateur du constructeur du parseur.
- `xsl:vendor-url` : retourne l'URL du constructeur du parseur.

string **format-number** (*number, string, string?*)

101

Permet de formater le nombre donné par le 1er argument à l'aide d'un format donné par le 2ème argument et le 3ème pour la partie décimale (syntaxe JDK1.1)

boolean **element-available** (*string*)

102

Retourne *true* si le *string* donné en paramètre est le nom d'une instruction reconnue par le parseur XSLT

boolean **function-available** (*string*)

103

Retourne *true* si le *string* donné en paramètre est le nom d'une fonction reconnue par le parseur XSLT.

Directement dans Netbeans mais aussi :

- XALAN <http://xml.apache.org/xalan-j/>
%java org.apache.xalan.xslt.Process -IN carnet.xml -XSL carnet.xsl -OUT carnet.txt
- Browser récent : Firefox, Chrome,
- Cocoon <http://xml.apache.org/cocoon>
- Saxon (Michael Kay) <http://saxon.sourceforge.net>
- XT (James Clark) <http://www.jclark.com/xt>
- Altovaxml.exe
- XmlSpy <http://www.altova.com>