le cnam

LES TECHNOLOGIES DU WEB

1ère Partie : **Introduction au Web**

- 1- **Introduction à l'Hypertexte**
- 2- **Présentation du protocole HTTP**
- 3- **Principes de bases des CGI**
- 4- **Présentation du WEB2** (AJAX)

2ème Partie : **Présentation de HTML & XHTML**

3ème Partie : **Présentation de Javascript**

4ème Partie : **Introduction à PHP**

5ème Partie : **Introduction à XML & XSLT**

le cnam

Une Introduction à XML

eXtensible Markup Language

le cnam

Plan du cours

- Introduction
  - ✓ Historique, principes, exemples
  - ✓ Comparaison avec HTML, SGML
  - ✓ Apports de XML
- La structure des documents XML
- Les grammaires
  - ✓ DTD et schémas XSD
- Les outils XML
  - ✓ XPath, XPointer, XLink
- Conclusion

le cnam

Objectifs

- Appréhender les concepts de base de XML
  - ✓ Eléments, attributs et contenu
  - ✓ DTD
  - ✓ Schémas
  - ✓ Espaces de noms
- Aperçu de deux recommandations annexes :
  - ✓ Les chemins de balise avec **XPath**
  - ✓ Les feuilles de style avec **XSL**

Historique

- 1986
  - ✓ SGML (*Standard Generalized Markup Language*)
  - ✓ Norme ISO:8879:1986
- 1987
  - ✓ TEI (*Text Encoding Initiative*)
- 1990
  - ✓ HTML 1.0 (*HyperText Markup Language*)
- 1997/1998
  - ✓ XML 1.0 (*eXtensible Markup Language*)
- 2000
  - ✓ XML 2.0 (*eXtensible Markup Language*)

Qu'est-ce que c'est ?

- Une Recommandation du W3C
- Un Langage de Balisage Extensible
- Un méta-langage
  - ✓ Permet de définir d'autres langages
- Une Simplification de la Norme SGML
  - ✓ ISO 8879:1986
  - ✓ Structure logique des documents électroniques
  - ✓ HTML est une application de SGML

Qu'est-ce que ça n'est pas ?

- Surtout pas une nouvelle version de HTML
  - ✓ HTML est un ensemble de balises figé, pour afficher du texte avec des liens hypertextes et des images
  - ✓ XML ne rend pas HTML obsolète (ni SGML !)
  - ✓ HTML devrait être compatible XML (XHTML)
- XML n'a pas de balise pré-définie et permet aux créateurs de spécifier leur propre jeu de balise pour structurer leurs données

Limites de HTML

- Une forte rigidité

  HTML permet de visualiser le contenu d'un document, mais de façon rigide
- Un manque d'extensibilité

  HTML ne propose qu'un nombre limité de balises
- Une montée en charge difficile

  HTML ne permet pas de travailler directement sur les informations contenues dans un document sans un accès au serveur
- Une interopérabilité limitée

  HTML ne fourni aucun moyen de décrire l'information échangée

  ➔ limite les tentatives de réutilisation

  HTML ne permet pas de partager des données entre plusieurs applications

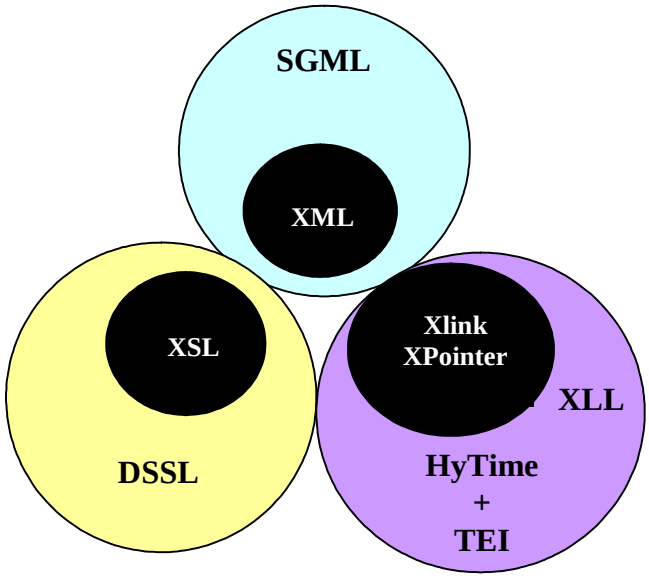
Apports de XML

- Vues multiples des données
  XML décrit les données, qui peuvent être affichées de différentes façons sur le poste client
- Traitement des données en local
  Les utilisateurs peuvent travailler directement sur les informations contenues dans un document sans un accès au serveur
- Une interopérabilité limitée
  Des données provenant de plusieurs sources peuvent être intégrées et manipulées par différentes applications
- Standards ouverts
  XML est défini par le W3C (XLink, Xpointer, XPath, XSLT, …)

Pour faire quoi ?

- Modélisation de structures de données
- Publication de données structurées sur le Web (documents mais pas seulement)
- Séparer la structure logique de l'affichage
- Applications distribuées
- Intégration de données en provenance d'applications hétéroclites

La famille XML



XSL : eXtensible Stylesheet Language

DSSL : Document Style Semantics and Specification Language

HyTime : Hypermedia Time-based structuring language

TEI : Text Encoding Initiative

Technologies liées à XML

- CSS, permet de définir une feuille de style pour XML.
- XSL, langage évolué pour la définition de feuilles de style.
- Xlink pour ajouter des liens hypertextes à un fichier XML.
- XPointer pour pointer sur des parties d'un document XML, un XPointer pointe sur des éléments de données au sein d'un fichier XML.
- DOM *Document Object Model* pour manipuler des fichiers XML (et HTML) à partir d'un langage de programmation.
- namespaces (domaines de noms) pour distinguer les noms utilisés dans les documents XML.
- XForm pour les formulaires.

le cnam

Applications XML (1)

- AML *Astronomical Markup Language* langage décrivant les différents types de données utilisées en astronomie.
- MathML *Mathematical Markup Language* notation mathématique sur le web.
- CML *Chemical Markup Language* pour la publication Internet des formules chimiques, de molécules, des équations, utilise une visionneuse Java nommée Jumbo pour visualiser les molécules.
- VML *Vector Markup Language* langage de balisage d'information graphique vectorielle.
- CDF *Channel Definition Format* utilisé par Microsoft pour décrire le contenu Active Channel.

le cnam

Applications XML (2)

- PGML *Precision Graphics Markup Language* décrit les structures de données graphiques complexes avec les primitives du langage Postscript. Il permet la conversion de documents aux formats ps et pdf en XML.
- SMIL *Synchronized Multimedia Integration Language* pour la création multimédia, il spécifie comment et quand des éléments multimédia peuvent apparaître dans une page web.
- RDF les applications traitant les données RDF peuvent récupérer les informations (auteur, URL, titre, description) et créer des bases de données permettant la recherche d'information.
- WML *Wireless Markup Language* le langage de balisage pour l'internet mobile.

XML plus en détail

- Considérez le document HTML suivant :

```
<html>
  <head>
        <title>Construire une application XML</title>
        </head>
  <body>
  <p>   jean-marc.pujos@cnam.fr <br>
  tél : 01 40 27 00 00 <br>
  fax : 01 40 27 00 10 <br>
  Département : Informatique</p>
</body>
</html>
```

  - ✓ Facile : une sorte de "fichier de signature"
  - ✓ Mais : le balisage décrit la manière dont l'information sera affichée par un *browser*
  - ✓ Parce que : aucune sémantique, aucune logique !...

En premier exemple de XML

- Considérez le document XML suivant :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE MEMBRE SYSTEM "http://…/MEMBRE.dtd">
<!-- Un membre du CNAM -->
<MEMBRE TYPE="IE" ID="M28">
  <NOM> PUJOS </NOM>
  <PRENOM> Jean-Marc </PRENOM>
  <MEL> jean-marc.pujos@cnam.fr </MEL>
  <TEL> 01 40 27 00 00 </TEL>
  <FAX> 01 40 27 00 10 </FAX>
  <EQUIPE LAB="Département">Informatique</EQUIPE>
</MEMBRE>
```

le cnam

Les avantages de XML

- Préserve la sémantique et la structure des données :
  - ✓ On pourrait utiliser un "script " pour accéder à l'adresse électronique du vingt huitième membre figurant dans un document XML

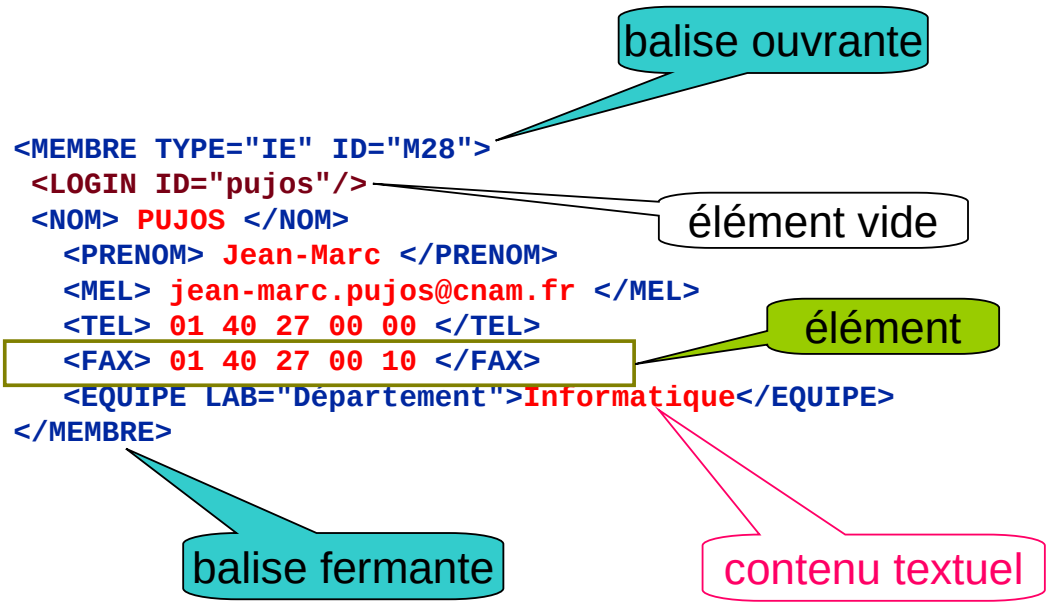
    `==> /DB/MEMBRE[28]/MEL/text()`
- Accent sur l'organisation des données
  - ✓ Sépare le contenu, de la structure et de la présentation

le cnam

Quelques propriétés de XML

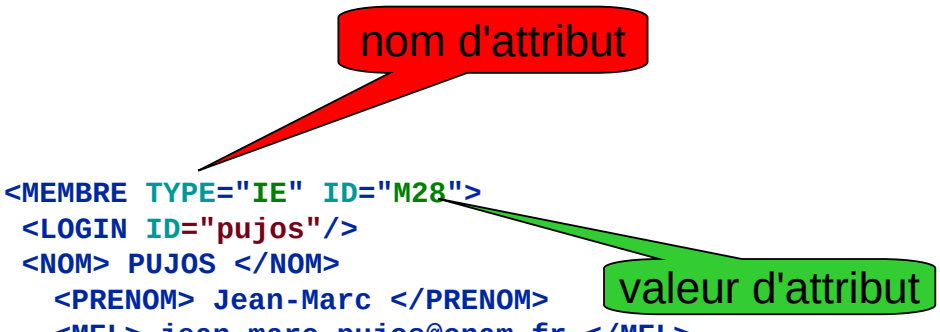
- Supporte les jeux de caractères **Unicode**
  - ✓ Attention minuscule ≠ majuscule
  - ✓ Les données peuvent contenir presque tous les caractères
- Les espaces en dehors du balisage sont par défaut, préservés
- Les chaînes littérales sont délimitées par des quotes simples ou doubles

Les éléments et leur contenu




```
<MEMBRE TYPE="IE" ID="M28">
 <LOGIN ID="pujos"/>
 <NOM> PUJOS </NOM>
   <PRENOM> Jean-Marc </PRENOM>
   <MEL> jean-marc.pujos@cnam.fr </MEL>
   <TEL> 01 40 27 00 00 </TEL>
   <FAX> 01 40 27 00 10 </FAX>
   <EQUIPE LAB="Département">Informatique</EQUIPE>
</MEMBRE>
```


Les éléments et leurs attributs




```
<MEMBRE TYPE="IE" ID="M28">
 <LOGIN ID="pujos"/>
 <NOM> PUJOS </NOM>
   <PRENOM> Jean-Marc </PRENOM>
   <MEL> jean-marc.pujos@cnam.fr </MEL>
   <TEL> 01 40 27 00 00 </TEL>
   <FAX> 01 40 27 00 10 </FAX>
   <EQUIPE LAB="Département">Informatique</EQUIPE>
</MEMBRE>
```

Autres éléments

- Déclaration XML

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

- Commentaires

```
<!-- ceci est un commentaire -->
```

- Section CDATA

```
<![CDATA[Langue & Dialogue]]>
```

- Instruction de traitement (pour une application)

```
<?edit line="wrap"?>
```


Structure d'un document XML (1)

Un document XML comporte des éléments avec ou sans attributs qui fournissent des méta-informations sur l'information ou sur le contenu du document.

Un document XML comporte :

- ✓ un prologue qui contient toutes les informations autres que les données ou les éléments,
- ✓ l'arbre des éléments avec un élément racine,
- ✓ éventuellement des commentaires.

Structure d'un document XML (2)

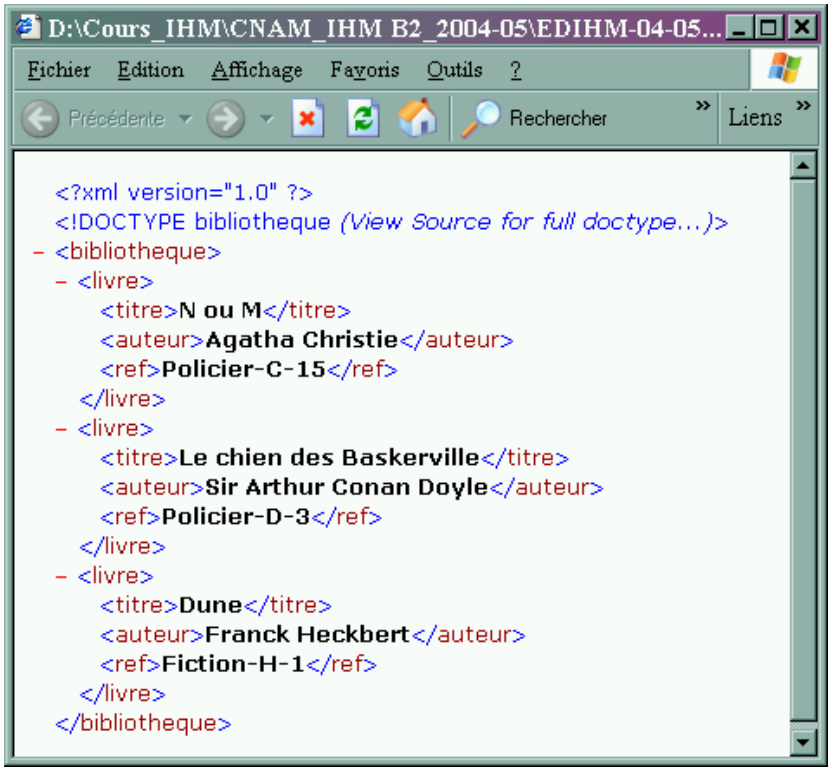

```
<?xml version="1.0"?>
<!DOCTYPE bibliotheque [       <!ELEMENT bibliotheque
(livre+)>
                    <!ELEMENT livre(titre,auteur, ref)>
                    <!ELEMENT titre (#PCDATA)>
                    <!ELEMENT auteur (#PCDATA)>
                    <!ELEMENT ref (#PCDATA)>
                     ]>
<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>
```

Prologue

Elément racine

Arbre d'éléments

Structure d'un document XML (3)



Besoin d'une ontologie

Document Type Definition / SCHEMA

- Définition de type de document
  - ✓ Contraintes sur les noms des éléments et des attributs
  - ✓ Occurrences des éléments et des attributs
  - ✓ Structure et organisation des éléments
- Approche SGML traditionnelle mais :
  - ✓ Optionnelle en XML et syntaxe simplifiée
  - ✓ Production valide et distribution bien-formée

➔ Ce sont les **DTD** *Document Type Definition* ou les schémas qui définissent les éléments et les règles d'utilisation (noms des éléments, attributs possibles pour un élément, imbrications). Cependant des documents XML peuvent ne pas avoir de DTD. Si un document a une DTD associée et qu'il se conforme à cette DTD, il est dit **valide**. S'il n'a pas de DTD et qu'il suit les règles définies par XML (par exemple : ses éléments sont correctement imbriqués) il est **bien formé**.

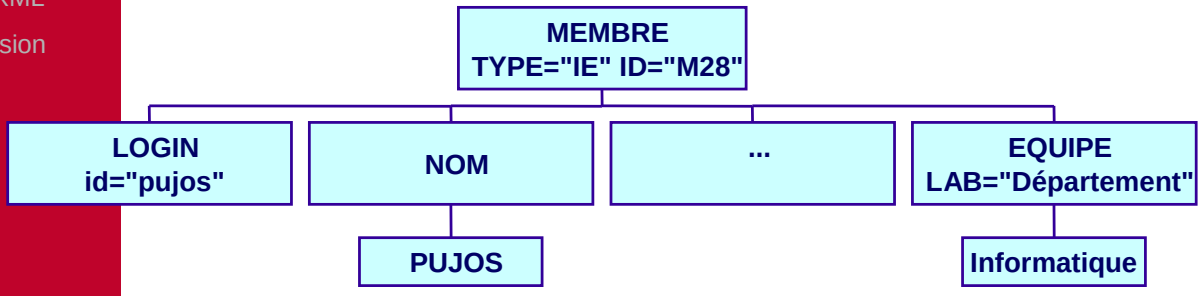
Document XML bien formé

Un document XML est bien formé (l'analyseur XML peut construire son arborescence) si :

- il contient une déclaration XML ;
- il contient un ou plusieurs éléments ;
- il contient un élément racine encapsulant tous les autres éléments et leurs attributs (ex `<HTML> ... </HTML>`) ;
- les éléments non vides ont une balise de début et de fin ;
- les éléments non vides sont correctement imbriqués (`<P> <EM> ... </EM> </P>`) ;
- les éléments vides ont un / à la fin de la balise avant le > ; **`<toto></toto> = <toto/>`**
- les noms des balises ouvrantes et fermantes correspondent ;
- un nom d'attribut apparaît uniquement dans la balise ouvrante et une seule fois dans cette balise ;
- les valeurs des attributs sont entre guillemets ou apostrophes ;
- la valeur des attributs n'appelle pas d'entités externes directement ou indirectement ;
- les caractères réservés sont remplacés par des références d'entités (par ex. `&lt;` pour <) ;
- toutes les références à des entités non binaires doivent commencer par & et finir par ;
- s'il n'y a pas de DTD, les seules entités utilisées sont celles réservées de XML `&amp; &lt; &gt; &apos; &quot;` ;
- s'il y a une DTD toutes les entités non réservées utilisées sont déclarées dans la DTD.

Arborescence XML

Une structure XML représente un arbre hiérarchique



Document XML valide

Un document est valide s'il :

- ✓ est bien formé,
- ✓ fait référence à une DTD ou à un schéma,
- ✓ se conforme à la DTD ou au schéma.

Les DTD *(Document Type Definition)* définissent les éléments et les règles d'utilisation : noms des éléments, attributs possibles pour un élément, imbrications (HTML 4.0 est une DTD de SGML).

Le document ne contient aucune information concernant l'affichage, c'est sa feuille de style qui définira la présentation sur un média.

Schéma XML

- Limite des DTD
  - ✓ description limité des documents structures
    - pas de typage des PCDATA
    - Limite
  - ✓ exprimé dans un langage autre XML
    - autre parseur, ...
- Objectif de XML Schéma
  - ✓ Amélioration du typage : xsi
    - données obéissant à une expression régulière
  - ✓ Amélioration de la structure : xsd
    - nombre borné d'éléments inclus, ...

➔ **Remplace de plus en plus la DTD car exprimé en syntaxe XML et extensible**

Les 2 parties de XML Schema

- Structure (Partie 1)
  - ✓ définit la construction de types (dit archetype)
    - simples ou complexes
    - nommés ou anonymes
    - mécanismes d'héritage
  - ✓ support pour les namespaces
    - validation de documents utilisant plusieurs namespaces
- Datatype (Partie 2)
  - ✓ définit un riche ensemble de types prédéfinies
  - ✓ incluant celui des langages de SGBD, ...

Exemple de schéma XML

Soit le fichier XML :

```
<?xml version="1.0" encoding="utf-8" ?>
<CATALOGUE>
 <PRODUIT NOM="T-shirt">
  <SPECIFICATION TAILLE="XL"
     COULEUR="BLANC"/>
  <PRIX GROS="5.00" DETAIL="12.00" TVA="2.35"
     EXPED="3.00"/>Euro</PRIX>
  <NOTES>Collection d'été</NOTES>
 </PRODUIT>
 <PRODUIT NOM="Chemise">
  <SPECIFICATION TAILLE="38"
     COULEUR="BLEUE"/>
  <PRIX GROS="11.00" DETAIL="35.00" TVA="6.86"
     EXPED="8.00">Euro</PRIX>
  <NOTES> </NOTES>
 </PRODUIT>
(autres produits)
</CATALOGUE>
```

Exemple de schéma XML (2)


```
<?xml version="1.0" encoding="utf-8" ?>
<catalogue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
si:schemaLocation="http://localhost/perso/MonExemple/catalog.xsd">
… La suite sans changement.
```

```
<?xml version="1.0" encoding="utf-8" ?>
    <xsd:schema targetNamespace="http://localhost/perso
    /MonExemple"
    "xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="produit">
    <xsd:complexType>
        <xsd:element name="specification">
        <xsd:complexType>
            <xsd:all>
            <xsd:attribut name="taille" type="xsd:string" />
            <xsd:attribut name="couleur" type="xsd:string" />
            </xsd:all>
        </xsd:complexType>
        </xsd:element>
    …
```


Exemple de schéma XML (3)


```
    …
    <xsd:element name="prix">
        <xsd:complexType>
            <xsd:all>
            <xsd:attribut name="gros" type="xsd:decimal" />
            <xsd:attribut name="detail" type="xsd:decimal" />
            <xsd:attribut name="tva" type="xsd:decimal" />
            </xsd:all>
        </xsd:complexType>
        </xsd:element>

     <xsd:element name="notes" minOccurs="0" type="xsd:string" />
    </xsd:complexType>
    </xsd:element>
    </xsd:schema>
```

Les Espaces de Noms XML

- Eviter les conflits de noms d'élément et d'attribut
  - ✓ Composition de fragments XML (BF)
  - ✓ Mixer, ré-utiliser plusieurs vocabulaires, schémas
  - ✓ Collection d'identificateurs (élément ou attribut) ***identifié par un préfixe et une URL***

- Déclaration :

```
      xmlns:mml="http://www.w3.org/Math/MathML/"
      xmlns="http://www.ua99.net/DOC/1.0">
  <P>blah blah :
      <mml:fn mml:definitionURL="mydef.xml">
      …
      </mml:fn> re blah blah</P>
</DOC>
```


Espaces de Noms Réservés

Le préfixe **`xml:`** est réservé pour certains attributs :

```
<title xml:space="default">...</title>
<p xml:lang="FR">…</p>
```

➔ On ne le déclare pas !…

L'attribut **`xml:lang`** a en plus la caractéristique d'être hérité. C'est à dire que tous les éléments (descendants) qui se trouvent sous un élément qui porte cette attribut hérite de sa valeur.

le cnam

XPath (1)

- XML Path Langage 1.0 REC 29012000
  - ✓ Langage permettant l'adressage de partie de documents XML
  - ✓ Spécification commune pour les pointeurs XML (**XPointer**) et le langage de transformation de XSL (**XSLT**)
  - ✓ Sélection, extraction de fragments XML
  - ✓ Noms d'éléments, noms/valeurs d'attributs

Les **XPath** résultent d'un vrai consensus entre le groupe de travail "feuilles de style" et le groupe de travail "liens et pointers". On dispose ainsi d'une syntaxe commune et d'une sémantique partagée.

le cnam

XPath (2)

- Recommandation du W3C
- Expression de désignation d'un noeud dans un document XML
  - ✓ syntaxe simple et non ambiguë
    - type usuels: chaînes, nombres, booléens, variables, fonctions
  - ✓ spécifie une bibliothèque de fonctions extensible
    - position(), …
- Exemples `/chapter[@type="warning"]`
  `//p[position()=5]`

Utilisé par XPointer et XSLT

L'Adressage avec les XPath

- Adressage absolu
  - ✓ A partir d'une **`URI/URL`**
  - ✓ **`id(M28)`**, **`here()`**, **`root()`**
- Adressage relatif en suivant des axes
  - ✓ **`ancestor`**, **`child`**
  - ✓ **`descendant`**
  - ✓ **`psibling`**, **`fsibling`**

Il s'agit d'un véritable langage de sélection de fragments de document XML. La maîtrise de ce langage est primordiale pour la conception de feuilles de style XSL.

Expressions XPath (1)

/	parenté section/paragraphe désigne les <paragraphe> fils des <section> de l'élément courant.
//	parenté étendu aux aïeux
.	noeud courant
..	père du noeud courant
|	alternative
@zzz	attribut zzz du noeud courant
text()	noeuds CDATA fils du noeud courant
comment()	noeuds commentaires fils du noeud courant
pi()	instructions de traitement du noeud courant
id()	sélection sur identifiant ou liste d'identifiants

Expressions XPath (2)

- Condition de sélection [ ]
  - ✓ section[@titre] <section> qui ont un attribut titre
  - ✓ section[paragraphe] <section> qui ont au moins un fils <paragraphe>
  - ✓ section[@titre='introduction'] <section> qui ont un attribut titre dont la valeur est "Introduction"
  - ✓ section[paragraphe='introduction'] <section> qui ont au moins un fils <paragraphe> dont la valeur est "Introduction«
- Positionnement
  - ✓ position()=5 vrai si l'élément est à la 5ème position
  - ✓ first-of-any() vrai si l'élément est le premier fils
  - ✓ last-of-any() vrai si l'élément est le dernier fils
  - ✓ first-of-type() vrai si l'élément est le premier fils de son type
  - ✓ last-of-type() vrai si l'élément est le dernier fils de son type

XPath - Exemples


```
<DB>
  <MEMBRE TYPE="IE" ID="M28">
    <LOGIN ID="pujos"/>
    ...
     <EQUIPE LAB="Département">Informatique</EQUIPE>
  </MEMBRE>
  <MEMBRE TYPE="CR" ID="M14">
    <LOGIN ID="dupont"/>
    ...
  </MEMBRE>
</DB>
```

/ ou /DB /DB/MEMBRE /DB/MEMBRE[2]

/DB/MEMBRE[@ID='M28']/EQUIPE[1]/text()

/DB/MEMBRE/LOGIN[@ID='dupont']/../@ID

eXtensible Linking Language - XLL

- Limite des références (URL) en HTML
  - ✓ ne référence qu'un seul document
  - ✓ granularité grosse de référence : le document
    - accèdes au 3ème paragraphe qui suit le chapitre nommé "Introduction"
  - ✓ pas de relations entre les documents.
  - ✓ référence à sens unique.
- Les 2 parties de XLL
  - ✓ Xlink : XML Linking Specification
  - ✓ XPointer : XML Extended Pointer Specification

Xpointer

XML Extended Pointer Specification :

- **adresse des objets internes à la structure du document et ne possédant pas d'identifiant**
  - ✓ Utilise les expressions Xpath
  - ✓ Exemple :

    http://www.ucc.ie/xml/faq.sgml#ID(faq-hypertext)CHILD(2,*)(6,*)

  - ➔ désigne le 6ème objet inclus dans le 2ème objet inclus dans l'élément ayant un ID égale « faq-hypertext ».
- **A lire**
  - ➔ *Recommandation XPointer du W3C et le chapitre 17 de « XML Bible »*

XLink

XML Linking Specification

Spécifie les hyperliens dans un document XML :

- ✓ Liens simples, type <A href="...">
- ✓ Liens étendus : multisource, multicible, externes
  - multi-direction, multi-cible, indépendance à la localisation (en cas de changement de place du document), transclusion (document inclus), liens typés (attributs de lien).

A lire

➔ *Recommandation XLink du W3C et le chapitre 16 de « XML Bible »*

XLink - Exemples

➢ Un lien simple :

```
<students xlink:href="students.xml"> The list of students.</students>
```

➢ Un lien étendu :

```
<element xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
      xlink:type="extended">
 <xlink:locator href="Source" role="role1"/>
 <xlink:locator href="Target" role="role2"/>
 <xlink:arc from="role1" to="role2" show="embed" actuate="auto"/>
 <xlink:title>The link title<xlink:title/>
 <xlink:title xml:lang="fr">Description du lien<xlink:title/>
   ...
</element>
```

➢ Annotations :

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
      role="xlink:external-linkset">
   <xlink:title>DV's Links</xlink:title>
   <xlink:locator href="http://rpmfind.net/veillard/linkset.xml"/>
</xlink:extended>
```

XML - Conclusion

- Étape révolutionnaire pour le Web
  - ✓ Consensus International (W3C)
- Cœxiste avec HTML et SGML
  - ✓ Ne supplante ni l'un ni l'autre
- Ouvre le *Web* à l'échange de données structurées et leur traitement

Une Introduction à XSLT

eXtensible Stylesheet Language Transformation

le cnam

Plan du cours

- Introduction
- Les Fondamentaux
- Structure d'une feuille XSL
- Principes de base
- Etude détaillée
  - ✓ gabarits, éléments, attributs, templates
- Programmation XSLT
  - ✓ tests, boucles, fonctions
- Exemples
- Conclusion

le cnam

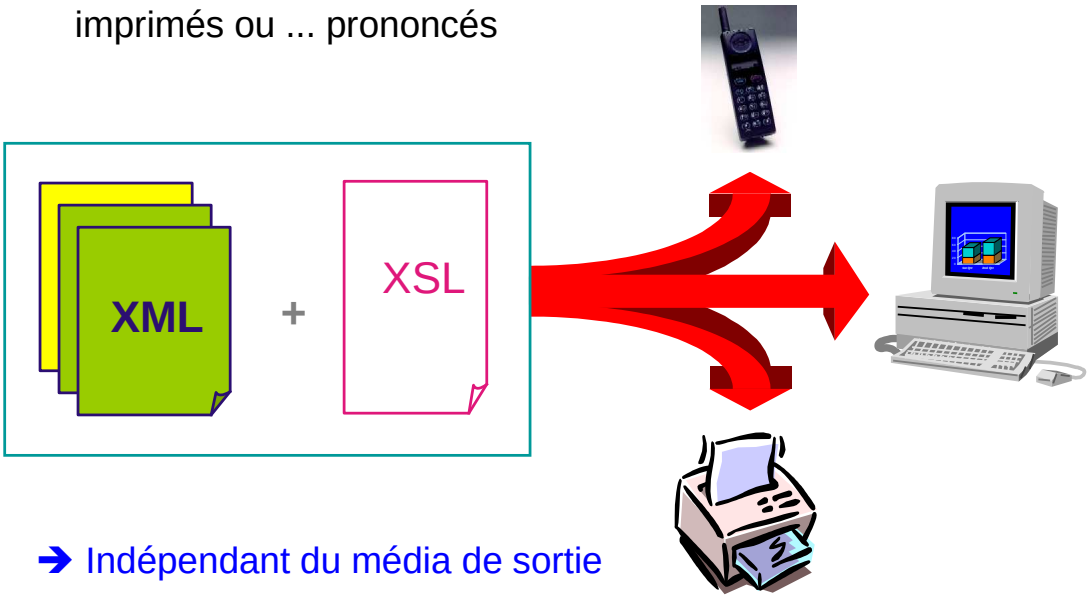
Pourquoi faire ?

Afficher des documents XML

- Un document XML ne fournit pas d'information sur sa présentation
- Affichage personnalisé
  - ➜ Feuilles de style :
    - *Casdading Style Sheets* (CSS 1 et 2)
    - *Extensible Style Language* (XSL)
- Transformation de documents XML

eXtensible Style Language

- Décrit la manière dont les documents XML seront affichés, imprimés ou ... prononcés

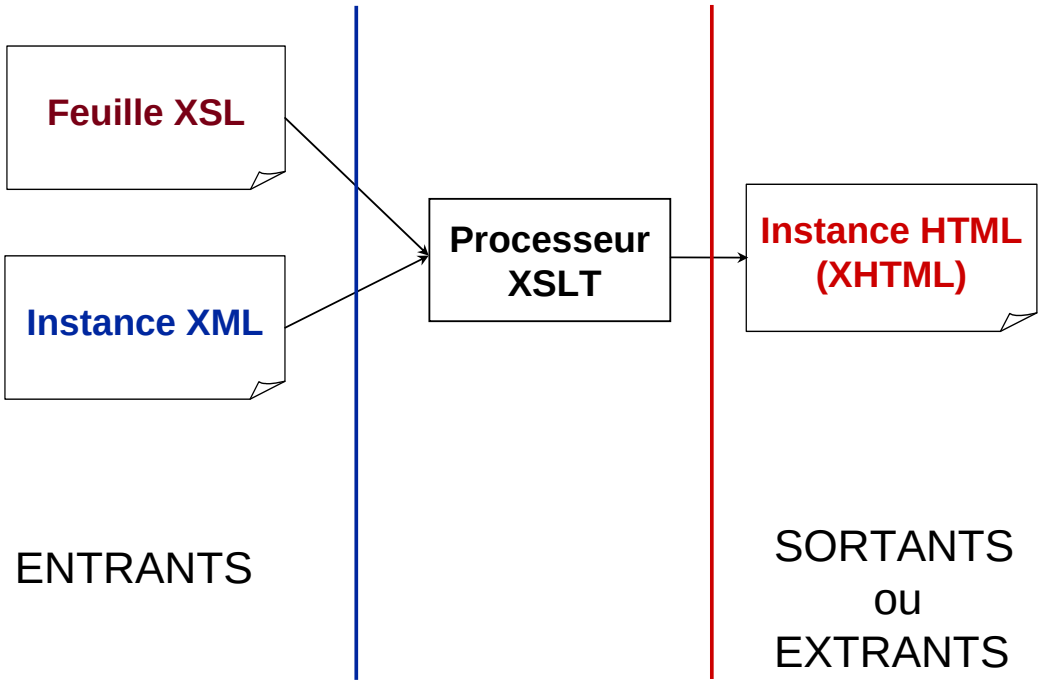


➔ Indépendant du média de sortie

Aspects fondamentaux de XSL

- XSL = Transformation + Propriétés d'Affichage
  - ✓ **XSLT : Transformation de documents XML**
    - **Transformer un doc XML en un autre doc XML**
    - **Par défaut : production de documents HTML (Bien formés !)**
  - ✓ **XSL FO : Formatage des données/objets XML**
    - **Les *Formatting Objects***
    - **Indépendant (Word/RTF, PS, PDF, MIF, …)**
- XSL comprend XSLT = XSL Transformations
- XSLT Version 1.0 est une recommandation W3C depuis le 16 novembre 1999
- XSLT utilise XPath (XML Path Language), aussi rec. W3C depuis le 16 novembre 1999

Transformation XSLT



Dans Internet Explorer

- Les navigateurs actuels incluent un processeur XSLT
- Une feuille de styles XSL a priorité sur une feuille CSS
- « Afficher source » affiche le XML
- « View XSL output » (du menu contextuel) affiche l'extrant de la transformation XSLT

Structure d'une feuille XSL

1- Prologue (exemple):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/2003/XSL/Transform">
```

2- Corps (suite de gabarits ou *templates*):

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

3- Epilogue:

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


Exemple de structure XSL


```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:template match="/">
  <html>
  <head>  <title>Ma bibliotheque</title>  </head>
  <body>
      <H2> Ma Bibliotheque</H2>
      <table border="1">
        <tr bgcolor="#FFFF00">
          <td>Titre</td>  <td>Auteur</td>  <td>Ref.</td>
        </tr>
        <xsl:for-each select="bibliotheque/livre">
        <tr>
          <td><div style="font-style:italic; padding-right:3pt"> <xsl:value-of
           select="titre"/></div>  </td>
          <td><div style="color:red; padding-right:3pt"> <xsl:value-of select="auteur"/></div>
          </td>
          <td><div style="color:blue"> <xsl:value-of select="ref"/></div> </td>
        </tr>
      </xsl:for-each>
      </table>
   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Prologue

Corps

Epilogue

Principes de base

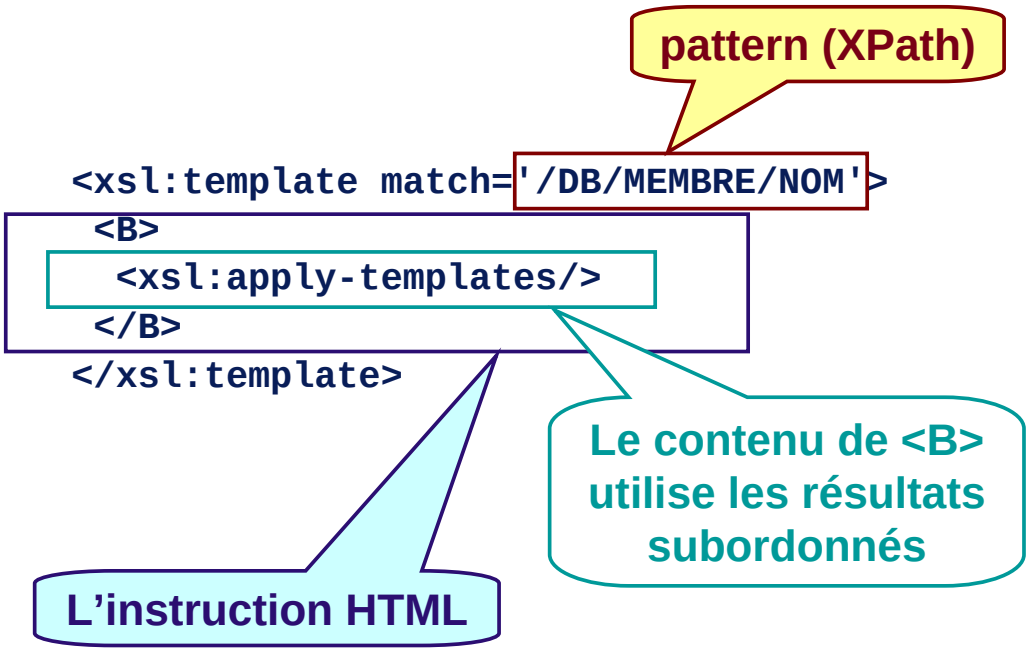
- Une feuille de styles XSLT est un document XML bien formé mais non valide
- Une transformation XSLT est une application non validante; elle est applicable à toute instance bien formée, valide ou non
- Une instruction de traitement:
  <?xml-stylesheet type="text/xsl" href="*mafeuille*.xsl" ?>

  dans le prologue de l'instance fait le lien avec la feuille de styles

Approche déclarative

- Suite de règles associées aux propriétés d'affichage qui spécifie à la fois le "motif" (*pattern*) et l'action de la règle
  - ✓ *pattern* : identifie les éléments du document source sur lesquels vont s'appliquer la règle
  - ✓ *template* : spécifie ce qu'on ajoute à l'arbre résultat lorsque la règle est activée
- Pas nécessaire d'utiliser les FO pour le résultat
- Les éléments HTML sont placés directement dans la feuille de style
  - ✓ Utilisation importante des Espaces de Noms XML
- Attention ! HTML n'est pas (encore) compatible XML
  - ✓ Les éléments vides XML **(`<br/>`)** peuvent poser des problèmes avec certains navigateurs

Exemple simple de règle



Notion de base : gabarit

- Gabarit = ***template*** = règle
- Chaque gabarit indique au processeur XSLT comment traiter certains éléments des instances dans certains contextes
- Exemple:

```
<xsl:template match="général">
  <p><b>Accessible au public</b></p>
</xsl:template>
```

Cas simple

match="*identificateur-générique*"

- S'applique à tous les éléments ayant cet identificateur générique
- Exemple:

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

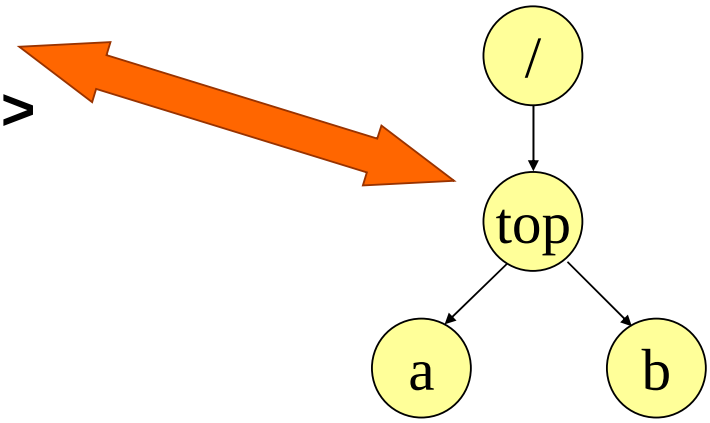

Exemple simple de `select` et de HTML


```
<xsl:template match='/'>
  <html>
   <head>
     <title>
      <xsl:apply-templates
      select='/DB/MEMBRE/NOM'/>
     </title>
   </head>
   <body>
    <xsl:apply-templates/>
   </body>
  </html>
</xsl:template>
```

Pseudo-élément racine

Le processeur XSLT considère que l'élément de plus haut niveau de l'instance a un parent, désigné par "/":

```
<?xml version = "1.0" ?>
<top>
  <a/><b/>
</top>
```



Action du processeur XSLT

- Regarde si (au moins) un des gabarits s'applique au pseudo-élément "/"
- Si oui, applique (ou exécute) ce gabarit
  - ✓ l'extrant de cette exécution de gabarit devient l'extrant de la transformation XSLT
- [Sinon, applique un gabarit fixe prédéfini]

Si on veut que d'autres gabarits soient exécutés, il faut qu'ils soient appelés, directement ou indirectement, par le gabarit appliqué à "/"

Extrant d'une exécution de gabarit

« L'extrant d'une exécution de gabarit est une chaîne de caractères égale au contenu du gabarit, dans lequel les "instructions XSLT" (s'il y en a) sont remplacées par l'extrant de leur exécution. »

Que peut contenir un gabarit :

- ✓ Des fragments HTML
  - **Ces fragments se retrouvent tels quels dans l'extrant du gabarit**
- ✓ Des instructions XSLT:
  - <xsl:apply-templates />
  - <xsl:apply-templates select="expression" />
  - <xsl:value-of select="expression" />
  - **etc.**

Gabarits prédéfinis

Des gabarits fixes prédéfinis existent pour:

- ✓ **Le pseudo-élément racine "/"**
- ✓ **Les éléments XML**
- ✓ **Les attributs**
- ✓ **Les nœuds de texte (#PCDATA ou String)**
- ✓ **etc.**

➢ Pour "/" et éléments, le gabarit prédéfini est:

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

➢ Pour noeuds textuels:

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

Priorité des gabarits

- Problème se pose lorsque plus d'un gabarit de la feuille de styles serait applicable à un même noeud
- Règles de priorité
- Si même priorité: dernier gabarit de la feuille est appliqué

Elément

- **Un élément non-XSLT qui apparaît dans un template crée un élément de même nom et avec les mêmes attributs (sauf attributs XSLT) dans l'arbre résultat**
- **On peut aussi créer des éléments dont le nom est engendré à l'exécution, avec l'instruction**

```
<xsl:element name="{attr-template}"
             use-attribute-sets="qnames">
  <!-- Content: template -->
</xsl:element>
```

  - ✓ **name : une expression XPath entre `{}` qui retourne une chaîne de caractères qui est le nom de l'élément à produire dans l'arbre résultat**
  - ✓ **use-attribute-sets : noms des ensembles d'attributs à engendrer pour l'élément**

Élément courant

- **"L'élément courant" dans un gabarit est l'élément pour lequel le gabarit est exécuté**
- **Exemple, avec:**

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

➔ **l'élément courant est un des éléments "auteur" de l'instance XML traitée**

Attribut

- On peut aussi créer des attributs dont le nom et la valeur sont engendrés à l'exécution, avec l'instruction

```
<xsl:attribute name="{attr-template}">
  <!-- Attribute value: template -->
</xsl:attribute>
```

- ✓ **name : une expression XPath entre** `{}` qui retourne une chaîne de caractères qui est le nom de l'attribut à produire dans l'arbre résultat
- ✓ **Le contenu de l'élément xsl:attribute est un template qui engendre la valeur de l'attribut**

Production de texte

- **On peut engendrer un noeud texte dynamiquement dans l'arbre résultat avec l'instruction**
  `<xsl:value-of select="string-expression"/>`
- **select : expression XPath de type chaîne qui produit la valeur du noeud texte**

Exemple **: élément source <personne prenom=" Pierre" nom="Paul"/>**

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

➔ **Résultat : `<p>Pierre Paul</p>`**

Expression

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

- **Extrait une chaîne de caractères du document, habituellement quelque part dans l'élément courant**
- **`select="."` retourne la valeur textuelle de l'élément courant**
- **`select="@attrib"` retourne la valeur d'un attribut de l'élément courant**
- **`select="IDGen"` retourne la valeur textuelle d'un (ou plusieurs) sous-éléments de l'élément courant**

le cnam

Templates (1)


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

- **Cause le traitement successif de chacun des "noeuds enfants" de l'élément courant par le gabarit approprié**
- **Un noeud enfant est soit un sous-élément, soit un noeud #PCDATA ou «*string*» (texte)**
- **L'ordre des enfants est respecté**
- **L'extrant de l'instruction est la concaténation des extrants résultant du traitement des enfants**

le cnam

Templates (2)


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

- **Fait traiter certains enfants spécifiques (répondant au critère de sélection)**
- **Cas simple: expression = identif. générique**
- **L'extrant de l'instruction est la concaténation des extrants résultant du traitement des enfants sélectionnés**
- **Permet de faire du "réarrangement" de sous-éléments**

le cnam

Bien-formé

- La feuille XSLT elle-même doit être bien-formée
- Donc, chaque gabarit doit être bien formé
- Donc, *tout élément HTML ouvert dans un gabarit doit être fermé dans le même gabarit*

D'ou la recommandation de bien écrire le code XHTML

le cnam

Notions avancées

- Il est possible de trier (`xsl:sort`)
- Les valeurs de `match=` et `select=` sont des expressions XPath
- En fait, XSLT est un langage de programmation complet

DEMONSTRATION

Programmer avec XSLT

XSLT offre de nombreuses fonctions dignes d'un langage de haut-niveau : variables, paramètres, tests, boucles, fonctions, inclusion d'une feuille de style XSLT dans une autre, chargement de plusieurs documents XML dans une même feuille de style XSLT, recherche de balises XML selon de nombreux critères, etc.

Fonctions XSLT

- xsl:for-each ➔ "Boucle"
- xsl:if ➔ Si conditionnel
- xsl:choose, xsl:when et xsl:otherwise
  ➔ Suite de codes conditionnels : instruction switch en C
- xsl:template, xsl:call-template, xsl:param et xsl:with-param ➔ Déclarer et appler une fonction, avec ou sans paramètre(s)
- xsl:number ➔ Numérotation/compteur

Boucles xsl:for-each

La fonction **xsl:for-each** va prendre tous les noeuds d'une requête XPATH, et va leur appliquer un traitement.

Exemple :

Le fichier source →

```
<liste>
  <invite>Moi</invite>
  <invite>Amélie</invite>
  <invite>Marie</invite>
  <invite>Jéméry</invite>
</liste>
```

Le code XSLT qui va générer le code HTML →

```
<ul>
<xsl:for-each>
  <li>
    <xsl:apply-templates select="." />
  </li>
</for-each>
</ul>
```


Boucles xsl:if

L'instruction **xsl:if** permet d'exécuter ou non certaines parties du code.

Exemple d'utilisation de la l'instruction if :

```
<xsl:if test="nom == 'Paul'">
 <xsl:text>Le spécialiste XML</xsl:text>
 </xsl:if>
```

le cnam

Boucles xsl:choose

La fonction **xsl:choose** permet d'exécuter différents codes selon différentes conditions.

Exemple :

```
<xsl:choose>
      <xsl:when test="nom = 'Paul'">
            <xsl:text>Le spécialiste XML</xsl:text>
            </xsl:when>
            <xsl:when test="nom = 'Pierre'">
            <xsl:text>L'infographiste</xsl:text>
      </xsl:when>
            <xsl:when test="nom = 'Pierre'">
                  <xsl:text>L'ergonome</xsl:text>
      </xsl:when>
</xsl:choose>
```


le cnam

Fonction sans paramètre

- Pour déclarer une fonction, on utilise la fonction **xsl:template**. Exemple :

```
<xsl:template name="hello_world">
  <xsl:text>Hello World !</xsl:text>
</xsl:template>
```

- Pour appeler la fonction, on utilise **xsl:call-template** :

```
<xsl:call-template name="hello_world" />
```

Fonction avec paramètres (1)

- Pour déclarer des paramètres, on utilise **xsl:param**

  Exemple :

```
<xsl:template name="affiche_somme">
  <xsl:param name="a" select="0" />
  <xsl:param name="b" select="0" />

  <xsl:text>a = </xsl:text>
  <xsl:value-of select="$a" />
  <xsl:text>, b = </xsl:text>
  <xsl:value-of select="$b" />
  <xsl:text>, et a+b = </xsl:text>
  <xsl:value-of select="$a + $b" />
  <xsl:text>.</xsl:text>
</xsl:template>
```


Fonction avec paramètres (2)

- Lorsqu'on écrit **<xsl:param name="a" select="0" />** : 0 est la valeur par défaut du paramètre a

  Appel de la fonction avec **xsl:call-template** :

```
<xsl:call-template name="affiche_somme">
  <xsl:with-param name="a" select="173" />
  <xsl:with-param name="b">9001</xsl:with-
param>
</xsl:call-template>
```

➔ ***Affichage obtenu :***

a = 173, b = 9001, et a+b = 9174.

Autres caractéristiques

XSLT offre de nombreuses autres possibilités :

- ✓ Contrôle du format de sortie (texte, HTML, XML) : élément xsl:output
- ✓ Numérotation : instruction xsl:number
- ✓ Tri : élément **`xsl:sort`**
- ✓ Variables et paramètres : éléments **`xsl:variable`**, **`xsl:param`**
- ✓ Accès à plusieurs documents source : fonction **`document`**
- ✓ Références croisées implicites : élément xsl:key, fonction key
- ✓ Messages : instruction **`xsl:message`**
- ✓ Contrôle des espaces : éléments **`xsl:strip-space`** et **`xsl:preserve-space`**

Exemple (1)

Document XML source (liste.xml) :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<liste_nombres>
  <nombre valeur="10">dix</nombre>
  <nombre valeur="0">zéro</nombre>
  <nombre valeur="33">trente trois</nombre>
  <nombre valeur="6">le premier nombre
parfait</nombre>
</liste_nombres>
```

Exemple (2)

Feuille de style XSLT (xslt.xsl) :

➔ pour afficher du HTML

```
<?xml version=« 2.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes"/>
<xsl:template match="liste_nombres">
  <html>
    <body>
      <p>Liste de nombres :</p>
        <ul>
              <xsl:apply-templates select="nombre" />
        </ul>
    </body>
  </html>
</xsl:template>
<xsl:template match="nombre">
  <li>
    <xsl:value-of select="@valeur"/>
    <xsl:text> : </xsl:text>
    <xsl:value-of select="."/>
  </li>
</xsl:template>
</xsl:stylesheet>
```

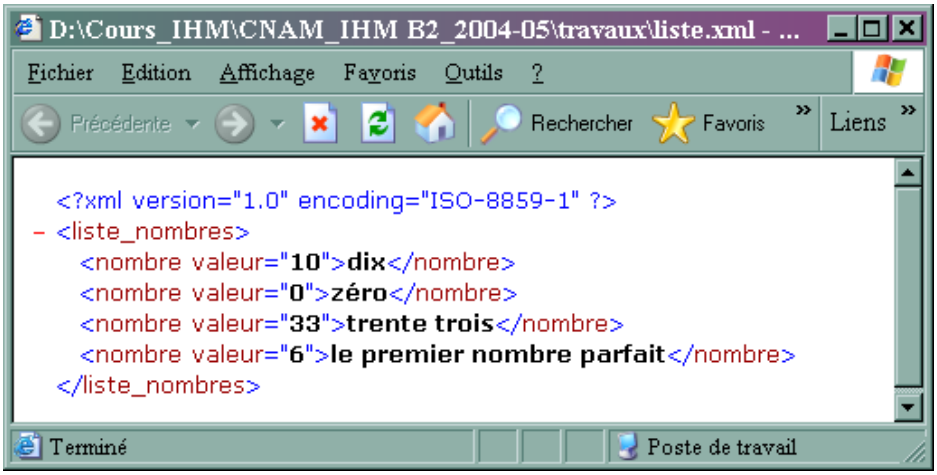

Exemple (3)

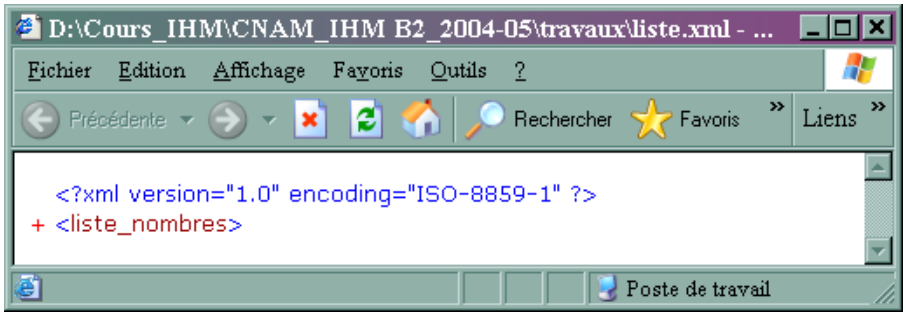
Appel au processeur XSLT en PHP :

```
<?
// Crée le processeur XSLT
$xh = xslt_create();
xslt_set_base ($xh, 'file://' . getcwd () . '/');

// Traite le document, puis affiche le résultat
$result = xslt_process($xh, 'liste.xml', 'xslt.xsl');

if (!$result)
  echo ("Erreur XSLT ...");
else
  echo ($result);

// Détruit le processeur XSLT
xslt_free($xh);
?>
```

le cnam

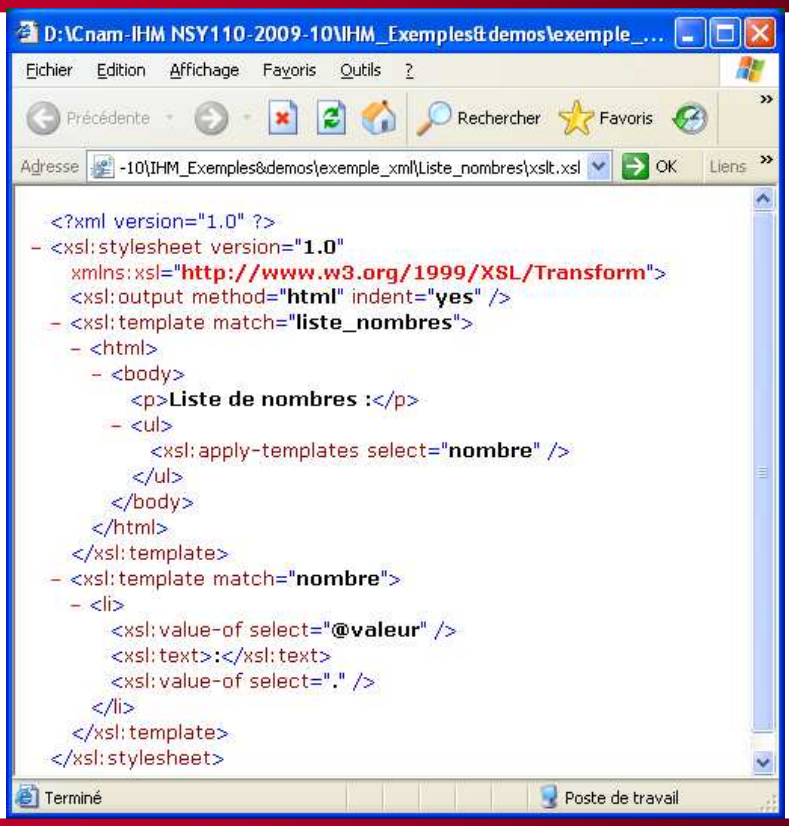
Visu du fichier "liste.xml" sans XSL





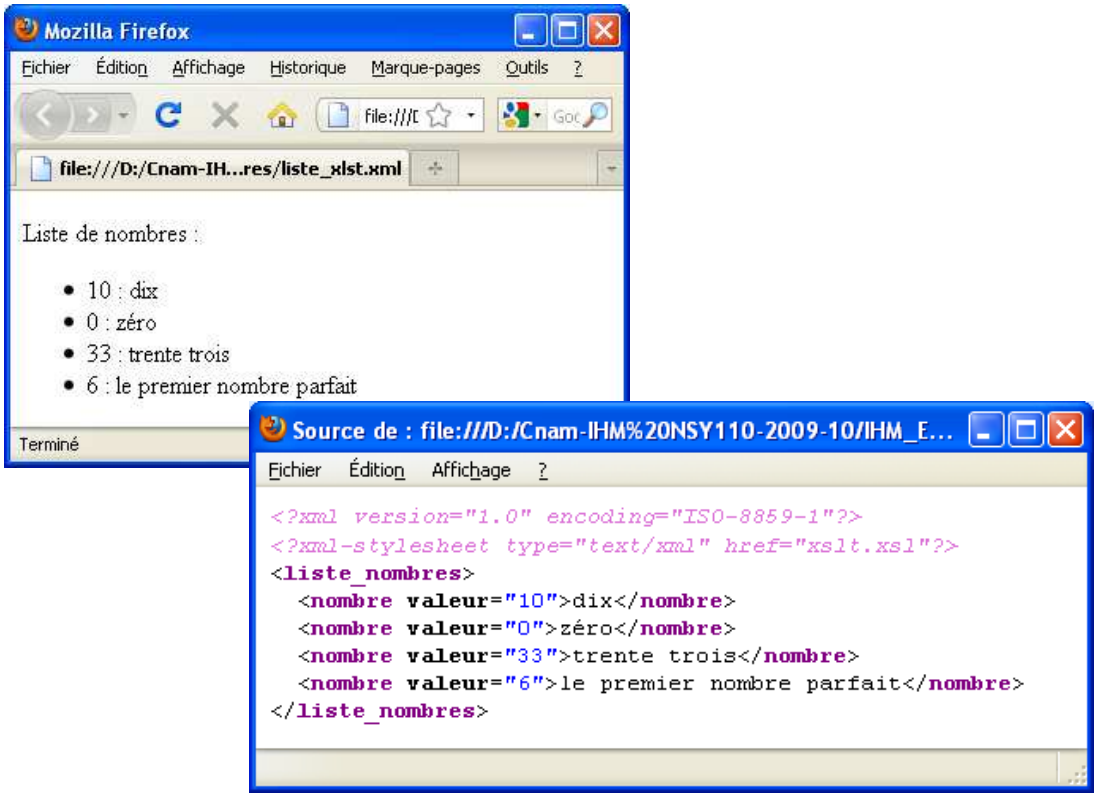
le cnam

Visu du fichier "xslt.xsl"



le cnam

Visu du fichier "liste.xml" avec XSL



le cnam

Conclusion

- XSLT : « le Perl de XML »
  Nombreuses implémentations (en particulier en open source), nombreux livres, XSLT propose de nombreuses extensions.
- Même si XSLT est puissant, il a des limitations :
  - ✓ Contrôle de structure ou de type (DTD, Schéma)
  - ✓ Traitement de documents streamés
  - ✓ Traitement incrémental
  - ✓ Transformations monotones
  - ✓ Composition de transformations
  - ✓ Réversibilité des transformations

le cnam

Références

- http://www.w3.org/XML/
- http://www.w3.org/TR
- http://www.w3.org/TR/REC-xml
- http://www.xml.com/
- http://www.xmlinfo.com/
- http://xml.apache.org/