

# **XML**

**M. DEYE**

L3 d'informatique

Université Cheikh Anta Diop de Dakar

# Pourquoi XML?

- Il existe plusieurs formats d'échange des données :
  - XML,
  - JSON,
  - YAML,
  - ...
- XML est le format le plus complet
- Connaître XML permet d'utiliser les autres sans difficulté

# PLAN (XML)

- **Présentation de XML**
- **La structure des documents XML**
- **La validation des documents XML**
- **L'extraction de fragments XML: XPath, Xquery**
- **La transformation des documents XML : XSLT**

# Présentation de XML

# Présentation de XML

- ❑ XML (*eXtended Markup Language*) est une spécification du W3C depuis février 1998, mais les premiers travaux autour de XML débutent dès 1996.
- ❑ XML n'est pas une nouveauté mais une succession d'un ensemble de technologies tels que le SGML (Standard Generalized Markup Language), HTML (HyperText Markup Language)
- ❑ Pourquoi XML ?
  - ❖ HTML n'est pas extensible, il ne peut pas répondre aux besoins spécifiques de tous les domaines (mathématiques, chimie, musique, astronomie...) et ne définit plus le contenu du document
  - ❖ SGML, qui permettrait de définir de nouveaux langages de balisage spécifiques, est complexe pour le web

# Présentation de XML

- ❑ Un langage (ou une méthode) permettant de structurer de l'information en employant un format texte comme support
- ❑ Utilisé pour échanger des données entre applications, stocker les préférences d'applications, créer des documents pour le web, ...

# Présentation de XML

- ❑ Un méta-langage = un langage qui permet de définir d'autres langages
- ❑ Permet de concevoir votre langage de balisage personnalisé qui est adapté pour décrire la structure particulière de vos données
- ❑ Le langage de balisage créé est généralement défini par une DTD ou un Schema XML

# Présentation de XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Deye</Nom>
    <Prénom>Mohamed</Prénom>
  </Auteur>
  <Description>Ce cours aborde les concepts de base de XML</Description>
</Cours>
```

C'est **presque** comme du HTML !

**Sauf que :**

- ☐ J'ai créé mes propres balises pour mes propres besoins
- ☐ Je suis contraint par une syntaxe rigoureuse qui n'accepte pas les erreurs

**Et que :**

- ☐ Ce document ne sait pas comment s'afficher dans un navigateur

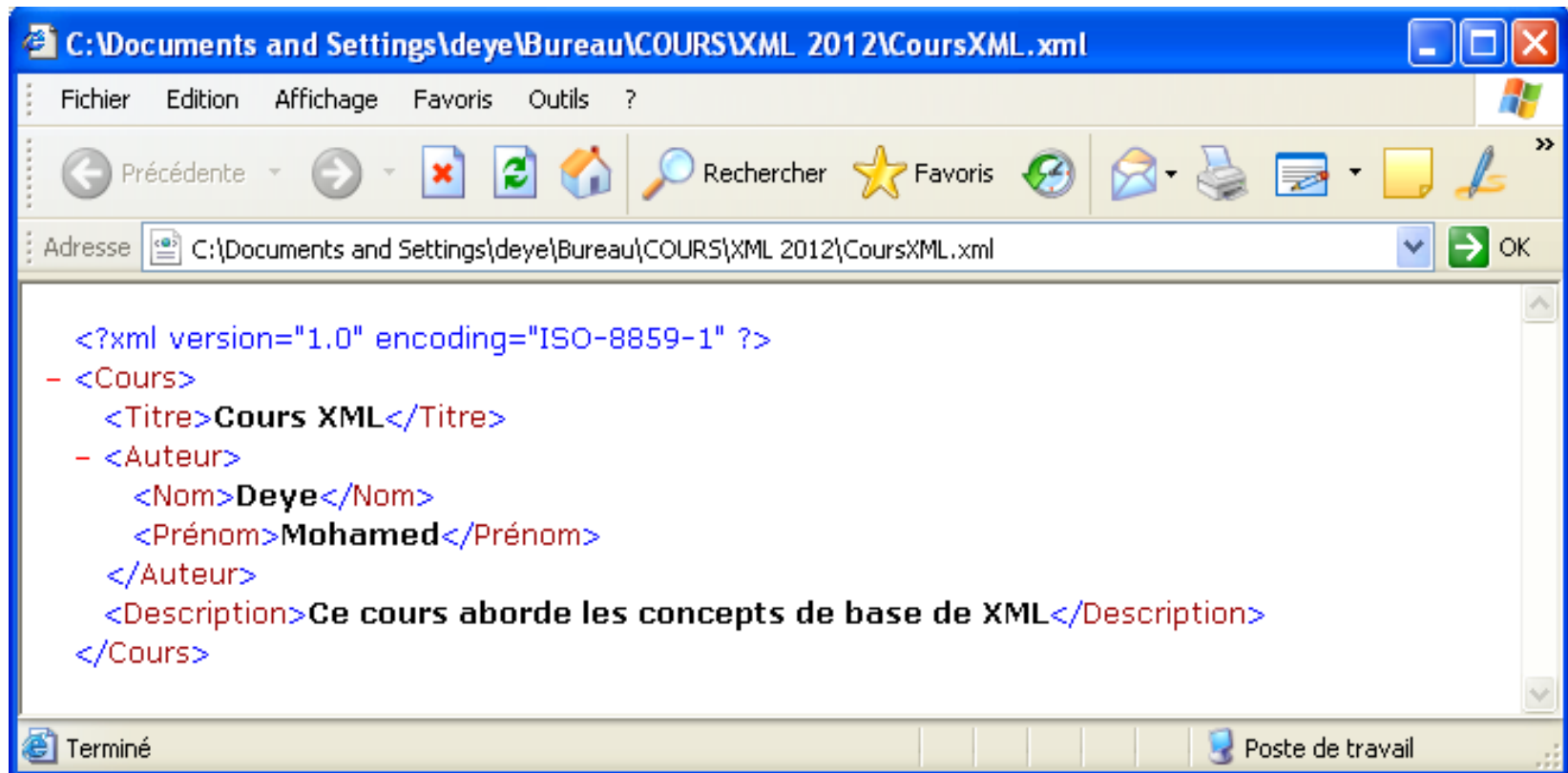
**Mais que :**

- ☐ Je dispose d'une boîte à outil pour manipuler les informations de ce document



# Présentation de XML

- ❑ Le XML, en lui-même, ne fait rien !
  - Il est destiné à décrire le contenu du document, pas son affichage (les feuilles de style CSS et XSL gèrent l'affichage)



The screenshot shows a Windows XP Notepad window titled "C:\Documents and Settings\deye\Bureau\COURS\XML 2012\CoursXML.xml". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", and "Outils". The toolbar contains icons for "Précédente", "Rechercher", "Favoris", and other standard Notepad functions. The address bar shows the file path "C:\Documents and Settings\deye\Bureau\COURS\XML 2012\CoursXML.xml". The main text area contains the following XML code:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Cours>
  <Titre>Cours XML</Titre>
  - <Auteur>
    <Nom>Deye</Nom>
    <Prénom>Mohamed</Prénom>
  </Auteur>
  <Description>Ce cours aborde les concepts de base de XML</Description>
</Cours>
```

The status bar at the bottom shows "Terminé" and "Poste de travail".

# Présentation de XML

## □ Un langage de balisages

- ❖ XML repose sur le balisage d'un flux de texte, comme HTML
- ❖ La différence fondamentale entre les deux langages consiste dans le fait qu'il n'existe aucune limitation quant aux balises de XML ! chacun peut inventer ses propres balises
- ❖ Une balise se symbolise de la façon suivante : `<nomdebalise>`
- ❖ Pour fermer une balise, on utilise : `</nomdebalise>`
- ❖ Utilise des balises pour décrire des données afin que d'autres applications (ou outils) puissent les lire et les traiter

# Présentation de XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Deye</Nom>
    <Prénom>Mohamed</Prénom>
  </Auteur>
  <Description>Ce cours aborde les concepts de base de XML</Description>
</Cours>
```

- ❑ Les balises sont des chaînes de caractères Unicode encadrées par les caractères "<" et ">"
- ❑ Entre une balise de début et la balise de fin correspondante, on peut trouver du texte ou d'autres balises

# Présentation de XML

- ❑ **Un document XML doit obligatoirement être bien formé :**
  - ✓ Tout document XML doit avoir un et un seul élément racine
  - ✓ Toute balise ouverte doit être fermée
  - ✓ Les balises doivent être correctement imbriquées entre elles
  - ✓ Les valeurs associées aux attributs doivent être encadrées par les délimiteurs « " » ou « ' »
  - ✓ Les noms des balises doivent commencer par une lettre ou "\_", les autres caractères peuvent être des chiffres, des lettres, "\_", "." ou "-".
  - ✓ Les noms des balises ne doivent pas commencer par xml
  - ✓ XML est key sensitive

# Présentation de XML

1

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

```
<personne>
```

```
  <nom>
```

```
  <prenom>
```

```
  </nom>
```

```
  </prenom>
```

```
</personne>
```

**Ko** : Mauvaise imbrication

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

```
<personne>
```

**Ko** : Balise non fermée

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

```
<personne/>
```

**Ok** : Élément vide

5

2

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

```
<personne>
```

```
</Personne>
```

**Ko** : Personne différent de personne

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

```
<xmlmine>
```

```
</xmlmine>
```

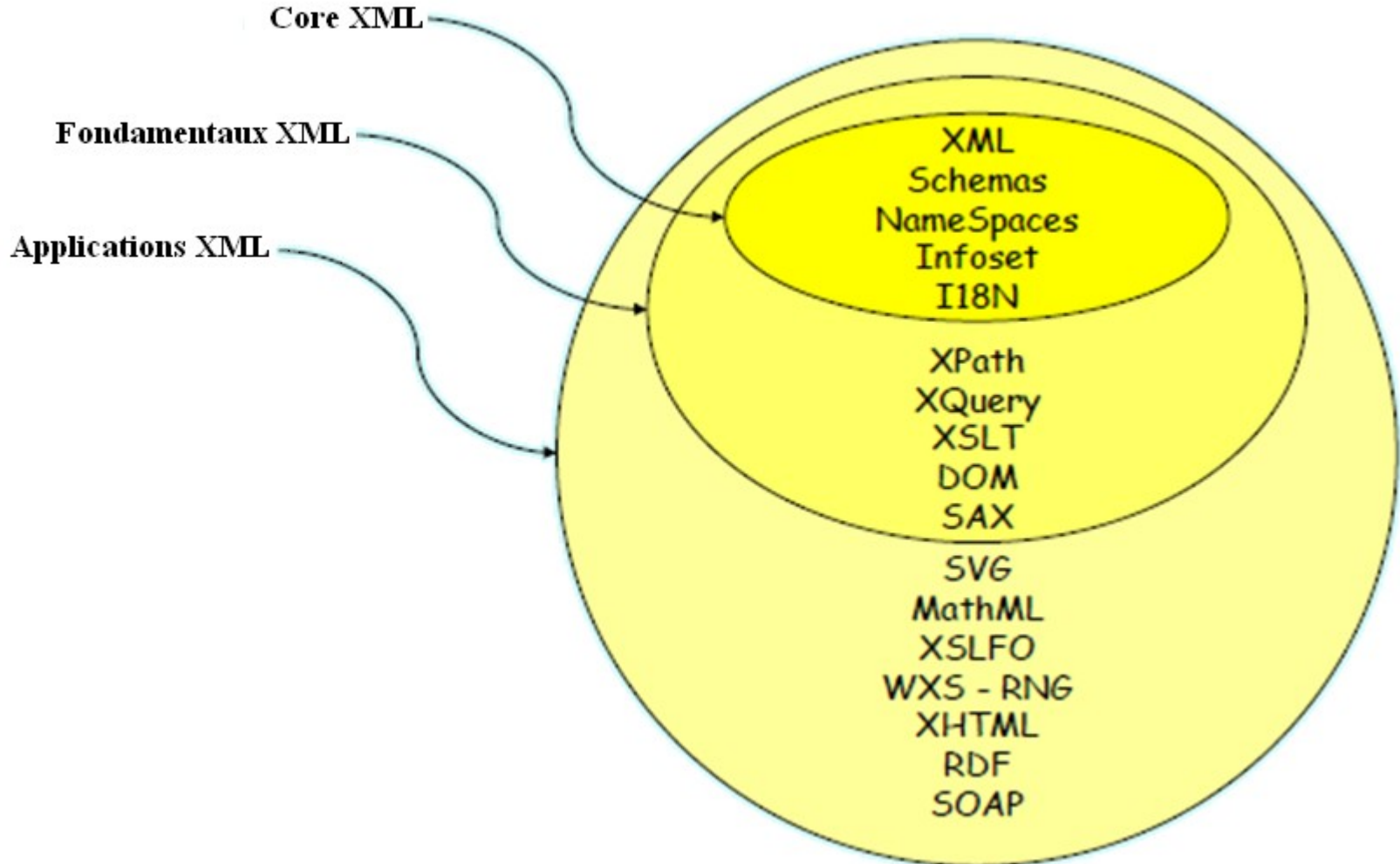
**Ko** : Pas de nom de balise commençant par xml

3

4

# Présentation de XML

## ☐ Technologies liées à XML



# Présentation de XML

- En fonction du contenu qu'on souhaite publier on définit ses propres balises. Quelques applications :
  - ✓ **MathML** *Mathematical Markup Language* notation mathématique sur le web
  - ✓ **SVG** *Scalable Vector Graphics* pour décrire des ensembles de graphiques vectoriels
  - ✓ **SMIL** *Synchronized Multimedia Integration Language* pour la création multimédia, il spécifie comment et quand des éléments multimédia peuvent apparaître dans une page web
  - ✓ **CML** *Chemical Markup Language* pour la publication Internet des formules chimiques, de molécules
  - ✓ ...etc

- Enregistrer dans un fichier avec extension “mml”
- Ensuite on l’ouvre dans le navigateur

```

<?xml version="1.0"?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>f</mi>
    <mo fence="true">(</mo>
    <mi>a</mi>    <mo>,</mo>    <mi>b</mi>    <mo
fence="true">)</mo>
    <mo>=</mo>    <mfrac>
      <mrow>
        <msup>
          <mi>a</mi>
          <mn>2</mn>
        </msup>
        <mo>+</mo>
        <mi>b</mi>
      </mrow>
      <msup>
        <mn>2</mn>
      </msup>
    </mfrac>
  </mrow>
</math>

```



- Enregistrer dans un fichier avec extension “svg”
- Ensuite on l’ouvre dans le navigateur

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

```
<svg xmlns="http://www.w3.org/2000/svg" width="467" height="462">
```

```
  <ellipse cx="150.23" cy="190.17" rx="250.45" ry="200.76" fill="green"  
stroke="none" />
```

```
  <circle cx="200.19" cy="350.45" r="150.71" stroke="orange" stroke-  
width="4.8" fill="yellow" fill-opacity="0.7" />
```

```
  <rect x="80.56" y="60.31" width="250.97" height="250.83" rx="20.2"  
fill="#ff0000" stroke="#000000" />
```

```
  <rect x="150.43" y="150.91" width="280.34" height="220.53" rx="40.98"  
style="fill:#0000ff; stroke:#000000; stroke-width:2.3px;fill-  
opacity:0.7;" />
```

```
</svg>
```

# Présentation de XML

- ❑ XML est largement diffusé pour décrire toutes sortes de structures arborescentes
- ❑ Citons par exemple :
  - ❖ Les documents contenant des ensembles de données (bons de commande, factures, dossiers, manuels techniques, etc.)
  - ❖ Les documents décrivant des structures à installer de façon répétitive (descripteurs de déploiement des applications J2EE)
- ❑ Par ailleurs, XML est bien adapté pour
  - ❖ L'échange de données entre des logiciels, c'est-à-dire dans les domaines de l'intégration des applications (EAI - Enterprise Application Integration) et dans le domaine de la communication interentreprises (B2B - Business to Business)

# Présentation de XML

- ❑ Les fichiers XML sont souvent des fichiers de grandes tailles contenant de très nombreuses informations reliées entre elles
- ❑ Ils sont généralement générés automatiquement par des programmes appropriés
- ❑ Les documents XML sont destinés essentiellement à être traités par des programmes informatiques appelés **parseurs XML**

# Présentation de XML

- ❑ Il existe des parseurs non validants qui n'offrent qu'une vérification syntaxique (bien formé ou non) et des parseurs validants qui offrent le support des DTD/schéma XML
- ❑ Deux méthodes de parsing :
  - ❖ Une méthode événementielle sans représentation globale du document : SAX (*Simple API for XML*)
  - ❖ Une méthode objet : DOM (Document Object Model)
- ❑ Editix, oXygen, ...
- ❑ La commande `xmllint` sous Linux

# La structure des documents XML

# La structure des documents XML

- ❑ Document : terme à prendre au sens large
- ❑ Un document XML peut exister physiquement d'un seul bloc mais dépend le plus souvent de l'assemblage de plusieurs unités physiques existant séparément et appelées entités.
- ❑ Les entités peuvent être schématisées comme des fichiers provenant de différents moyens de stockage situés à différents emplacements
- ❑ Chaque entité peut elle-même appeler d'autres entités

# La structure des documents XML

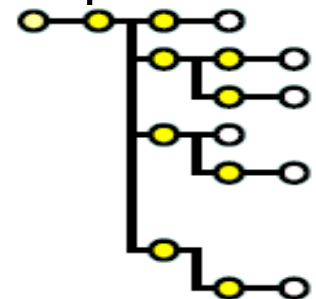
## ❑ La structure physique d'un document XML

- ❖ Séquence de caractères "à plat"
- ❖ Spécifie l'encodage caractère
- ❖ Spécifie le découpage en fichiers
- ❖ Peut contenir des caractères non significatifs (indentations)

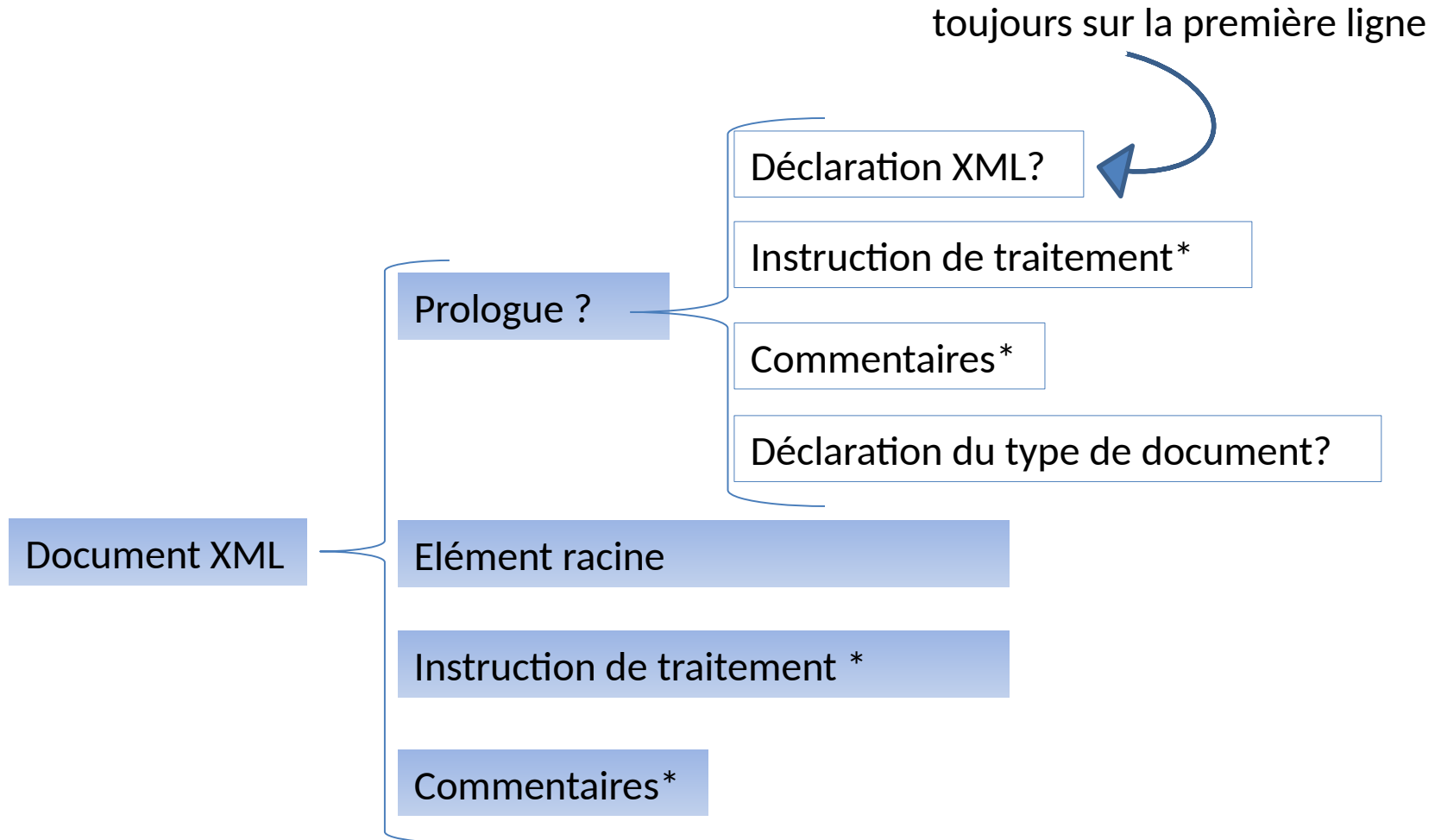
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Ould Deye</Nom>
    <Prénom>Mohamed Mahmoud</Prénom>
  </Auteur>
  <Description>Ce cours aborde les concepts de base de XML</Description>
</Cours>
```

## ❑ La structure logique d'un document XML

- ❖ Une arborescence d'informations obtenue par un processeur XML à partir du contenu physique



# La structure des documents XML





# La structure des documents XML

## ❑ La déclaration XML

- ❖ n'est pas obligatoire
- ❖ Si elle est présente, elle doit apparaître en premier et débute par "**<?xml**" et se termine par "**?>**"

## ❑ Elle fournit trois informations sous la forme de pseudo-attributs

- ❖ **version** : cet attribut est obligatoire quand la déclaration XML est présente **<?xml version="1.0" ?>**
- ❖ **encoding** : cet attribut indique le type de codage que doivent utiliser les processeurs XML afin de traduire des octets en caractères à la réception d'un document, ou de transformer des caractères en données binaires pour le transport . Il permet à des applications de se comprendre
- ❖ **<?xml version="1.0" encoding="ISO-8859-1" ?>**

# La structure des documents XML

## ❑ La déclaration XML : encoding

Norme	Correspondance
UTF-8	Jeu de caractères universel sur 8 bits
UTF-16	Jeu de caractères universel sur 16 bits
Norme	Correspondance
ISO-8859-1	Latin 1 – Langues d'Europe de l'ouest et d'Amérique latine
ISO-8859-2	Latin 2 – Langues d'Europe centrale et Slaves
ISO-8859-3	Latin 3 – Langues Espéranto, Galicienne, Maltaise et Turc
ISO-8859-4	Latin 4 – Langues Estonienne, Lettonne et Lithuanienne
ISO-8859-5	Langue Cyrilliques
ISO-8859-6	Langue Arabe
ISO-8859-7	Langue Grecque
ISO-8859-8	Langue Hébraïque
ISO-8859-9	Latin 5 – Langue Turc
ISO-8859-10	Latin 6 – Langues Groenlandaises et Laponnes

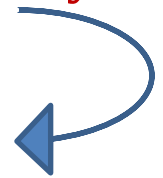
# La structure des documents XML

## ❑ La déclaration XML

- ❖ **standalone** : cet attribut indique la nature autonome ou non d'un document.
- ❖ Un document est déclaré non-autonome lorsque l'entité document qui le représente est dépendante de déclarations de balisage externes

**<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>**

- Documents sans DTD
- Documents avec DTD internes



# La structure des documents XML

## ❑ Instructions de traitement (Processing Instructions)

- ❖ Instructions spéciales Destinées à une application spécifique

**<?cible arg1 arg2 ... ?>**

- ❖ cible : nom de l'application
- ❖ arg1, arg2 : arguments passés à l'application

**<?xml-stylesheet type="text/css" href="style.css" ?>**

- ❖ Cette instruction indique au navigateur d'afficher les données XML en appliquant la feuille de style : "style.css"

## ❑ Commentaires (Comment)

- ❖ peuvent être placés dans un fichier XML ou un fichier DTD

**<!-- commentaire -->**

# La structure des documents XML

- ❑ **Déclaration du type de document** (Document Type Declaration)
  - ❖ n'est obligatoire que pour la validation
  - ❖ Si elle est présente, elle doit se trouver dans le préambule
  - ❖ Un document qui déclare une DTD doit s'y conformer
  
- ❑ **<!DOCTYPE** racine **SYSTEM** "URI vers la DTD" **>**
  - ❖ Exemple : **<!DOCTYPE** cours **SYSTEM** "cours.dtd" **>**
  
- ❑ **<!DOCTYPE** racine **PUBLIC** "identifiant\_public" "url"? **>**
  - ❖ PUBLIC est utilisé lorsque la DTD est une norme ou qu'elle est enregistrée sous forme de norme ISO par l'auteur
  - ❖ Exemple : **<!DOCTYPE** html **PUBLIC** "-//W3C//DTD XHTML 1.0 Strict//FR" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" **>**

# La structure des documents XML

## ❑ **Élément racine**

- ❖ C'est l'élément (ou la balise) qui contient tous les autres éléments

## ❑ **Un élément** ( une balise) a un nom : une balise ouvrante et une balise fermante

## ❑ Un élément peut contenir :

- ❖ **Rien** : il n'y pas de contenu, l'élément est vide.
- ❖ **Du texte**
- ❖ **Un ou plusieurs éléments**
- ❖ **Un mélange de textes et d'éléments** : une forme plus rare mais reste utile, lorsque l'on souhaite « décorer » un texte quelconque (cas du paragraphe en HTML avec des zones en gras, italique...).

# La structure des documents XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
```

```
<!DOCTYPE cours SYSTEM "cours.dtd">
```

```
<cours>
```

```
  <intervenant>Deye</intervenant>
```

```
  <separateur/>
```

```
  <chapitre>
```

```
    XML et Les Services Web
```

```
    <para>Un paragraphe</para>
```

```
    <para>Autre paragraphe</para>
```

```
  </chapitre>
```

```
</cours>
```

# La structure des documents XML

## ❑ Les attributs d'un élément :

❖ Un attribut est un couple (clé, valeur) associé à la définition d'un élément. Il est localisé dans la balise ouvrante de l'élément

❖ Exemple :

**<intervenant mail="deye@ucad.sn" >Deye</intervenant>**

❖ mail est un attribut de l'élément intervenant



# La structure des documents XML

## ❑ Les nœuds textes :

❖ Dans un document XML, ce qui est appelé donnée est le texte qui est associé à l'attribut, c'est-à-dire sa valeur, ou à l'élément, c'est-à-dire son contenu.

❖ Exemple :

**<calcul>**

**if ( a<b et b>c)**

**</calcul>**

❖ on voit bien que <b et b> n'est pas une balise mais fait partie des données mais pour un parseur c'est une balise de syntaxe incorrecte

❖ Solution =====> utilisation d'**entités prédéfinies**

# La structure des documents XML

## ❑ Les nœuds textes :

❖ Voici la liste des entités prédéfinies :

**&lt;** équivalent de <

**&gt;** équivalent de >

**&amp;** équivalent de &

**&quot;** équivalent de "

**&apos;** équivalent de ‘

❖ L'exemple précédent peut donc être correctement réécrit :

**<calcul>**

**If (a&lt;b et b&gt;c)**

**</calcul>**

# La structure des documents XML

## ❑ Les nœuds textes :

- ❖ Les entités prédéfinies ne servent qu'à lever une ambiguïté syntaxique pour le parseur. Elles peuvent être utilisées dans un élément ou dans un attribut
- ❖ Ces entités présentes en trop grand nombre dans un même bloc peuvent alourdir inutilement le document
- ❖ Pour le contenu textuel d'un élément (et seulement pour le contenu d'un élément), nous disposons des sections **CDATA** (Character Data)

# La structure des documents XML

## ❑ Les nœuds textes :

❖ Une section **CDATA** est considérée comme un bloc de texte dont les caractères seront pris tel quel par le parseur jusqu'à la séquence de fin **]]>**.

❖ Exemple :

```
<![CDATA[  
<element>C'est un document XML  
</element>  
]]>
```

❖ **<element>** n'est pas considéré comme une balise de structuration, mais comme du texte.

# La structure des documents XML

## ❑ Les espaces de noms :

- ❖ L'importation d'éléments ou d'attributs contenus dans des entités externes peut entraîner des conflits de noms
- ❖ Ces conflits peuvent être évités en définissant des espaces de noms
- ❖ Les espaces de noms en XML, c'est un moyen
  - Pour identifier de façon universelle les noms utilisés dans les documents
  - Et éviter les conflits de noms
- ❖ Un espace de noms est une collection de noms d'éléments ou noms d'attributs (**identifiée par un URI**)

# La structure des documents XML

- ❑ **Notion d'URI (Uniform Resource Identifiers) :**
  - ❖ Permettent d'identifier les espaces de noms
  - ❖ Doivent être persistantes et universellement uniques
  - ❖ Structure des URI : Structure générique  
**[procédure:]information-spécifique[#fragment]**
  - ❖ des modèles conformes à cette structure : les **URL** et les **URN**
  - ❖ URL : <http://www.masociete.com>
    - La partie procédure correspond à un protocole
    - L'information-spécifique indique un nom d'hôte

# La structure des documents XML

## ❑ Déclaration d'un espace de noms :

- ❖ La déclaration d'un espace de noms et de son préfixe associé consiste à insérer dans la balise ouvrante d'un élément contenant des noms (d'éléments ou d'attributs) issus de cet espace, l'attribut :

**xmlns:préfixe="URI de l'espace de noms"**

- ❖ On peut déclarer un espace de noms par défaut par l'attribut :

**xmlns="URI de l'espace de noms"**

# La structure des documents XML

## ❑ L'espace de noms par défaut :

- ❖ Il est précisé par un pseudo-attribut **xmlns**. La valeur associée sera une URL garantissant l'unicité de l'espace
- ❖ L'espace par défaut s'applique à l'élément où se situe sa déclaration et à tout son contenu
- ❖ Exemple :

**<chapitre xmlns="http://www.masociete.com">**

**<paragraphe>**

**...**

**</paragraphe>**

**</chapitre>**



# La structure des documents XML

## ❑ L'espace de noms explicite :

- ❖ Il se déclare à l'aide d'un préfixe qui est défini comme un pseudo-attribut commençant par **xmlns:prefixe**

- ❖ Exemple :

```
<p:chapitre xmlns:p="http://www.masociete.com">  
  <p:paragraphe>...</p:paragraphe>  
</p:chapitre>
```

- ❖ Lorsqu'un élément est préfixé, son contenu doit l'être aussi si l'on souhaite que l'espace de noms s'applique également

# La structure des documents XML

## ❑ Exemples :

```
<?xml version="1.0" encoding=" ISO-8859-1" ?>
<dossier_scolaire>
  <dénomination xmlns="http://ucad.sn/etudiants"> Abdou Gaye</dénomination>
  <dénomination xmlns="http://ucad.sn/diplomes"> Mastère </dénomination>
  <dénomination xmlns="http://ucad.sn/matieres"> Informatique </dénomination>
</dossier_scolaire>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <title>Ma page</title>
  </head>
  <body>
    <p>Mon texte</p>
  </body>
</html>
```

# La validation des documents XML

# La validation d'un document XML

- Les échanges de données dans l'entreprise sont divers.
- Certaines données sont liées à une opération manuelle et d'autres à une opération automatique mais dans les deux cas, des erreurs sont toujours possibles.
- La validation est un moyen pour vérifier que votre document est conforme à une grammaire.
- Les DTD (Document Type Definition)
- Les XML Schema

# La validation d'un document XML

- Une DTD est une sorte d'entité contenant une description formelle de la structure et du vocabulaire (noms de type d'éléments et noms d'attributs) d'un document.
- La spécification XML définit complètement l'écriture des DTD, mais une DTD n'est pas elle-même un document XML
- Un parseur validateur lit la DTD avant de lire le document.

# La validation d'un document XML

- Permet de définir le "vocabulaire" et la structure qui seront utilisés dans le document XML
- Grammaire du langage dont les phrases sont des documents XML (instances)
- Peut être mise dans un fichier et être appelée dans le document XML (DTD externe)
- Pauvre en possibilités de contrôle (typage de données, par exemple)

# La validation d'un document XML

- **DTD interne**

- Les déclarations de la DTD sont placés dans le DOCTYPE après la déclaration XML

- L'attribut **standalone** a la valeur "yes"

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

`<!DOCTYPE document [déclarations]>`

`<document > ... </document >`

- **document** : nom du type de document déclaré
- **déclarations** : déclarations d'éléments, d'attributs et d'entités...

# La validation d'un document XML

- **DTD externe**

- Les déclarations sont placés dans un fichier séparé portant l'extension .dtd

**<!DOCTYPE document (SYSTEM "uri")>**

- L'attribut **standalone** a la valeur **"no"**

**<?xml version="1.0" standalone="no" >**

**<!DOCTYPE document SYSTEM  
"http://ucad.sn/document.dtd">**

**<document > ... </document >**

- un chemin relatif :

**<?xml version="1.0" standalone="no" >**

**<!DOCTYPE document SYSTEM "document.dtd">**

**<document > ... </document >**



# La validation d'un document XML

- **DTD externe**

- Le mot PUBLIC est utilisé lorsque la DTD est une norme : **<!DOCTYPE document (PUBLIC "identifiant\_public" "url"?)>**
- Exemple : xhtml
  - <?xml version="1.0" encoding="ISO-8859-1"?>**
  - <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">**
  - <html>**
  - ...**
  - </html>**

# La validation d'un document XML

- **3 types de déclaration de DTD**

- On déclare la DTD et on y ajoute les définitions dans le même fichier (**DTD interne**)

**<!DOCTYPE document [déclarations]>**

- On déclare la DTD en tant que DTD "privée", la DTD se trouve quelque part dans votre système ou sur Internet.

**<!DOCTYPE document (SYSTEM "uri")>**

- On déclare une DTD "public", c.a.d. on utilise un nom officiel pour la DTD.

**<!DOCTYPE document (PUBLIC "identifiant\_public" "url"?)>**

# La validation d'un document XML

- **Déclaration d'un élément**

- **<!ELEMENT balise Def\_Contenu>**
- Décrit une balise **balise** qui fera partie du vocabulaire
- **Def\_Contenu** peut contenir :
  - **EMPTY** : l'élément n'a pas de contenu. Il peut cependant avoir des attributs.
  - **ANY** : l'élément peut contenir n'importe quel élément présent dans la DTD.
  - **(#PCDATA)** : l'élément contient du texte.
  - **Un élément** placé entre parenthèses comme **(nom\_element)**.
  - **Un ensemble d'éléments** séparés par des opérateurs, le tout placé entre parenthèses. L'opérateur de choix, représenté par le caractère | et l'opérateur de séquence, représenté par le caractère ,.

# La validation d'un document XML

- **Exemple :**

**<!ELEMENT personne (nom\_prenom | nom)>**

**<!ELEMENT nom\_prenom (#PCDATA)>**

**<!ELEMENT nom (#PCDATA)>**

– Cela nous autorise deux documents XML, soit :

**<personne>**

**<nom\_prenom>Brillant Alexandre</nom\_prenom>**

**</personne>**

– ou bien :

**<personne>**

**<nom>Brillant</nom>**

**</personne>**

# La validation d'un document XML

- **Exemple :**

**<!ELEMENT personne(prenom,nom)>**

**<!ELEMENT prenom (#PCDATA)>**

**<!ELEMENT nom (#PCDATA)>**

- L'opérateur de séquence limite les possibilités à un seul document XML valide :

**<personne>**

**<prenom>Alexandre</prenom>**

**<nom>Brillant</nom>**

**</personne>**

# La validation d'un document XML

- **Déclaration d'un élément** : Les contenus (élément ou groupe d'éléments) peuvent être quantifiés par les opérateurs \*,+ et ?.

- Opérateurs d'occurrence

- **Notations**

- (a, b) séquence
    - (a | b) liste de choix
    - a? élément optionnel [0,1]
    - a\* élément répétitif [0,N]
    - a+ élément répétitif [1,N]

- Exemples**

- (nom, prenom, rue, ville)
    - (oui | non)
    - (nom, prenom?, rue, ville)
    - (produit\*, client)
    - (produit\*, vendeur+)

# La validation d'un document XML

- **Exemples :**

**<!ELEMENT chapitre (auteur\*,paragraphe+)>**

- L'élément **chapitre** contient de 0 à n éléments **auteur** suivi d'au moins un élément **paragraphe**.

**<!ELEMENT livre (auteur?,chapitre)+>**

- L'élément **livre** contient au moins un élément, chaque élément, étant un groupe d'éléments où l'élément **auteur**, est optionnel et l'élément **chapitre** est présent en un seul exemplaire.

# La validation d'un document XML

- **Exemple :**
  - Supposons, La description des livres d'une librairie
    - Stocker des informations sur titre, auteur, éditeur, prix
    - C'est possible d'avoir plusieurs auteurs pour un livre
    - Le prix est optionnel
    - Les prix seront classés par catégorie
    - cat1 pour les livres dont le prix est inférieur à 20€
    - cat2 pour les livres dont le prix est entre 20€ et 40€
    - cat3 pour les livres dont le prix est supérieur à 40€
    - cat1, cat2 et cat3 sont des éléments vides



# La validation d'un document XML

- Nœud racine est **librairie**
- `<!DOCTYPE librairie [  
... instructions ...`
- `]>`

# La validation d'un document XML

- Dans notre librairie, on peut avoir 0 ou plusieurs livres
- `<!DOCTYPE librairie [  
    <!ELEMENT librairie (livre*)>  
    ]>`

# La validation d'un document XML

- L'information d'un livre (titre, auteur, éditeur, prix)
- `<!DOCTYPE librairie [  
 <!ELEMENT librairie (livre*)>  
 <!ELEMENT livre(titre, auteur, éditeur,  
 prix)>  
 ]>`

# La validation d'un document XML

- Un livre peut avoir **un ou plusieurs auteurs**
- `<!DOCTYPE librairie [  
 <!ELEMENT librairie (livre*)>  
 <!ELEMENT livre(titre, auteur+, éditeur,  
prix)>  
 ]>`

# La validation d'un document XML

- Le prix d'un livre est **optionnel**
- `<!DOCTYPE librairie [  
 <!ELEMENT librairie (livre*)>  
 <!ELEMENT livre(titre, auteur+, éditeur,  
 prix?)>  
 ]>`

# La validation d'un document XML

- Titre, auteur et éditeur sont stockés comme chaîne de caractère.
- `<!DOCTYPE librairie [  
 <!ELEMENT librairie (livre*)>  
 <!ELEMENT livre(titre, auteur+, éditeur, prix?)>  
 <!ELEMENT titre (#PCDATA)>  
 <!ELEMENT auteur (#PCDATA)>  
 <!ELEMENT éditeur (#PCDATA)>  
 ]>`

# La validation d'un document XML

- Prix sont stockés soit **cat1, cat2 ou cat3**
- **<!DOCTYPE librairie [**  
  **<!ELEMENT librairie (livre\*)>**  
  **<!ELEMENT livre(titre, auteur+, éditeur, prix?)>**  
  **<!ELEMENT titre (#PCDATA)>**  
  **<!ELEMENT auteur (#PCDATA)>**  
  **<!ELEMENT éditeur (#PCDATA)>**  
  **<!ELEMENT prix (cat1 |cat2 | cat3)>**  
• **]>**

# La validation d'un document XML

- Cat1, cat2, cat3 sont des éléments qui sont **sans valeurs**
- `<!DOCTYPE librairie [  
 <!ELEMENT librairie (livre*)>  
 <!ELEMENT livre(titre, auteur+, éditeur, prix?)>  
 <!ELEMENT titre (#PCDATA)>  
 <!ELEMENT auteur (#PCDATA)>  
 <!ELEMENT éditeur (#PCDATA)>  
 <!ELEMENT prix (cat1 |cat2 | cat3)>  
 <!ELEMENT cat1 EMPTY)>  
 <!ELEMENT cat2 EMPTY) >  
 <!ELEMENT cat3 EMPTY)>  
 ]>`



# La validation d'un document XML

- **La définition d'un attribut**

- Les attributs sont précisés dans l'instruction ATTLIST.

**<!ATTLIST balise attribut type valeur\_par\_defaut>**

- Le type peut être principalement :
    - **CDATA** : du texte (Character Data) ;
    - **ID** : un identifiant unique;
    - **IDREF** : une référence vers un ID ;
    - **IDREFS** : une liste de références vers des ID ;
    - **NMTOKEN** : un mot (donc pas de blanc) ;
    - **NMTOKENS** : une liste de mots (séparation par un blanc);
    - **Une énumération de valeurs** : chaque valeur est séparée par le caractère |.

# La validation d'un document XML

- **La définition d'un attribut**
  - La valeur par défaut peut être :
    - **"Valeur"** : une valeur donnée.
    - **#REQUIRED** : attribut obligatoire.
    - **#IMPLIED** : attribut optionnel.
    - **#FIXED valeur** : l'attribut prend toujours cette valeur.

# La validation d'un document XML

- **Exemples :**

**<!ATTLIST chapitre titre CDATA #REQUIRED  
auteur CDATA #IMPLIED>**

- L'élément chapitre possède ici un attribut titre obligatoire et un attribut auteur optionnel.

**<!ATTLIST crayon couleur (rouge|vert|bleu) "bleu">**

- L'élément crayon possède un attribut couleur dont les valeurs font partie de l'ensemble rouge, vert, bleu.

# La validation d'un document XML

- **Exemple** : liens hypertextuels internes : ID ou IDREF
  - Dans la DTD

**<!ELEMENT SECTION (#PCDATA|xref)\*>**

**<!ATTLIST SECTION TARGET ID #IMPLIED>**

**<!ELEMENT xref EMPTY>**

**<!ATTLIST xref ref IDREF #REQUIRED>**

- Dans l'instance

**<SECTION TARGET="cible"> contenu </SECTION>**

**<SECTION>**

**référence à la section**

**<xref ref="cible"/>**

**</SECTION>**

# La validation d'un document XML

- **Exemple :**

- Le prix d'un livre est **optionnel** et **c'est l'attribut** de l'élément livre qui a des valeurs, soit **cat1**, **cat2** ou **cat 3**

- `<!DOCTYPE librairie [`

- `<!ELEMENT librairie (livre*)>`

- `<!ELEMENT livre(titre, auteur+, éditeur)>`

- `...`

- `<!ATTLIST livre prix(cat1|cat2|cat3) #IMPLIED>`

- `]>`

# La validation d'un document XML

- **Les entités :**
  - Une entité est un fragment nommé d'un document
  - Ces sont des « raccourcis » vers des portions de documents allant du caractère au document complet
  - Ces entités doivent être définies dans l'en-tête du document XML, ou dans la DTD, et peuvent être référencées à une ou plusieurs reprises dans le document.
  - Pourquoi utiliser des entités ?
    - Pour gagner du temps dans les mises à jour
    - Pour mémoriser des caractères particuliers ou des phrases fréquentes

# La validation d'un document XML

- **Types d'entités**
  - **Entités prédéfinies** : caractères réservés de XML
  - **Entités caractères** : caractères UNICODE désignés par leur code numérique
  - **Entités générales** : fragments nommés de l'élément du document
  - **Entités paramètres** : fragments nommés de DTD

# La validation d'un document XML

- **Appel d'entité**

- Un appel d'entité a l'une des 3 formes suivantes :
  - **&#code;** : appel de caractère
  - **&nom;** : appel d'entité générale ou prédéfinie
  - **%nom;** : appel d'entité paramètre
- Un appel d'entité prédéfinie, caractère ou générale ne peut apparaître que dans la valeur d'un attribut ou dans le contenu d'un élément.
- Un appel d'entité paramètre ne peut apparaître que dans la DTD.



# La validation d'un document XML

- **La définition d'une entité**

- Une entité associe un nom à une valeur qui peut être interne ou externe
- Pour la forme interne : **<!ENTITY nom "VALEUR" >**
- Dans la forme externe, on se retrouve avec le même principe qu'avec l'instruction **DOCTYPE**
- Les mots-clés **SYSTEM** et **PUBLIC** servent à réaliser un lien vers une valeur présente dans un fichier

# La validation d'un document XML

- **Déclaration d'une entité générale :**
  - interne
    - **<!ENTITY *nom* "*entité*">**
  - externe
    - **<!ENTITY *nom* SYSTEM "*nom du fichier contenant l'entité*">**

# La validation d'un document XML

- **Déclaration d'une entité paramètre :**
  - interne
    - **<!ENTITY % *nom* "*entité*">**
  - externe
    - **<!ENTITY % *nom* SYSTEM "*nom du fichier contenant l'entité*">**

# La validation d'un document XML

- **Exemple :**

```
<?xml version="1.0"?>
<!DOCTYPE démo [
  <!ELEMENT démo (#PCDATA | important)>
  <!ELEMENT important (#PCDATA)>
  <!ENTITY % identificateur "ID #REQUIRED">
  <!ENTITY cie "Les aliments de Dakar SA">
  <!ENTITY imp "<important>Attention!</important>">
  <!ATTLIST démo cible %identificateur;>
]>
<démo cible="dkr">
  &cie;
  &imp;
</démo>
```

# La validation d'un document XML

- **Exemple :**

```
<?xml version="1.0"?>
<!DOCTYPE doc [
    <!ENTITY chap1 SYSTEM "../chapitres/chap1.xml">
    <!ENTITY chap2 SYSTEM "../chapitres/chap2.xml">
]>
<doc>
    &chap1;
    &chap2;
</doc>
```

# La validation d'un document XML

- **Entités caractères**

- Un caractère non disponible sur la station de travail peut être représenté par son code Unicode en décimal ou en hexadécimal, sous la forme d'une référence à une entité :

- *&#code décimal;*
- *&#xcode hexadécimal;*

- **Par exemple :**

- **&#38;** le caractère &
- **&#x03A6;** la lettre grecque  $\Phi$
- © s'écrit **&#169;** ou **&#xA9;**

# La validation d'un document XML

- **Entités prédéfinies**

- Les caractères **<** **>** **&** **'** **"** qui sont des délimiteurs XML doivent être remplacés dans un texte par les entités suivantes :

- **&lt;** réfère le caractère **<**
    - **&gt;** réfère le caractère **>**
    - **&amp;** réfère le caractère **&**
    - **&apos;** réfère le caractère **'**
    - **&quot;** réfère le caractère **"**

# XML Schéma

- XML Schéma est une recommandation du W3C de mai 2001 qui consiste à définir une méthodologie de description de document XML autrement que par les DTD.
- L'approche cherche à résoudre plusieurs problèmes des DTD dont l'épine dorsale est le fait qu'un DTD n'est pas un document XML.
- En conclusion, il est nécessaire de connaître un autre langage que XML pour décrire un DTD (le langage propre aux DTD) mais également de disposer d'autres outils pour éditer et manipuler des DTD (par exemple, des parseurs spéciaux).



# XML Schéma

- Un XML Schéma est donc un document dont l'objectif est similaire à un DTD mais sous forme d'un document en XML.
- Plus clairement, c'est un document XML qui décrit un document XML !
- C'est pour cette raison qu'on parle de méta données pour les données constituant un XML Schéma.
- XML Schema est un document XML qui utilise un ensemble de balises définies au sein de la recommandation du W3C.

# XML Schéma

- Le but d'un schéma est de définir une classe de documents XML.
- Il permet de décrire les autorisations d'imbrication et l'ordre d'apparition des éléments et de leurs attributs, tout comme une DTD. Mais il permet aussi d'aller au-delà.
- Un premier point intéressant est qu'un fichier Schema XML est un document XML
- Cela permet à un tel document d'être manipulé de la même manière que n'importe quel autre fichier XML

# XML Schéma

- Les DTD présentent l'inconvénient d'être pauvre en possibilités de contrôle (typage de données, par exemple).
- Dans une DTD toutes les définitions sont globales. Cela signifie qu'il n'est pas possible d'utiliser plusieurs définitions pour un même nom d'élément.

**<document>**

**<titre>...</titre>**

**<auteur>**

**<titre>...</titre>**

**</auteur>**

**</document>**

il faudrait décrire toutes les possibilités de contenu pour l'élément titre, avec la conséquence que l'on ne pourrait contrôler l'usage de telle ou telle possibilité (par exemple, le titre de l'auteur employé à la place du titre du document).

# Structure de base

- Comme tout document XML, un Schema XML commence par un prologue, et a un élément racine.

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">  
  
  <!-- déclarations d'éléments, d'attributs et de types ici -->  
  
</xsd:schema>
```

# Déclarations d'éléments

- Un élément, dans un schéma, se déclare avec la balise **<xsd:element>**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="contacts" type="typeContacts" />
  <xsd:element name="remarque" type="xsd:string" />
  <!-- déclarations de types ici -->
</xsd:schema>
```

Chaque élément déclaré est associé à un type de données via l'attribut type. Les éléments pouvant contenir des élément-enfants ou posséder des attributs sont dits de type *complexe*, tandis que les éléments n'en contenant pas sont dits de type *simple*

# Déclarations d'attributs

- Contrairement aux éléments, un attribut ne peut être que de type simple.
- Cela signifie que les attributs, comme avec les DTD, ne peuvent contenir d'autres éléments ou attributs.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="contacts" type="typeContacts" />
  <xsd:element name="remarque" type="xsd:string">
    <xsd:complexType>
      <xsd:attribute name="datecom" type="xsd:date" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

# Contraintes d'occurrences

- Tout comme dans une DTD, un attribut peut avoir un indicateur d'occurrences.
- L'élément **attribute** d'un Schema XML peut avoir trois attributs optionnels : **use**, **default** et **fixed**.
- Des combinaisons de ces trois attributs permettent de paramétrer ce qui est acceptable ou non dans le fichier XML final (attribut obligatoire, optionnel, possédant une valeur par défaut...).

```
<xsd:attribute name="datecom" type="xsd:date" use="optional"  
default="2003-10-11" />
```

# Contraintes d'occurrences

- **Déclaration d'une valeur par défaut**
  - `<xsd:attribute name="lang" type="xsd:string" default="EN"/>`
- **Déclaration d'une valeur fixe**
  - `<xsd:attribute name="lang" type="xsd:string" fixed="EN"/>`
- **Déclaration d'un attribut optionnel**
  - `<xsd:attribute name="lang" type="xsd:string" use="optional" />`
- **Déclaration d'un attribut obligatoire**
  - `<xsd:attribute name="lang" type="xsd:string" use="required"/>`



# Les types de données communs

- XML schema a beaucoup de types de données. Voici la liste des types les plus commun
  - xsd:string
  - xsd:decimal
  - xsd:integer
  - xsd:boolean
  - xsd:date
  - xsd:time

# La création de nouveaux types simples

Les types simples peuvent être étendus pour créer de nouveaux types :

- **La restriction** : en limitant certaines caractéristiques (ex. la longueur d'une chaîne);
- **L'union** : autoriser plusieurs types, ce qui permet d'offrir des alternatives;
- **La liste** : il s'agit d'un ensemble de valeurs de même type séparées par un blanc.

# Restriction de type simple

- Avec les chaînes :
  - **xsd:length** : longueur en nombre de caractères ;
  - **xsd:maxLength** : longueur maximale ;
  - **xsd:minLength** : longueur minimale.
- Pour les nombres et les dates :
  - **xsd:maxExclusive** / **xsd:maxInclusive** : borne supérieure ;
  - **xsd:minExclusive** / **xsd:minInclusive** : borne inférieure.
- Pour les nombres seulement :
  - **xsd:totalDigits** : nombre de chiffres d'un entier ;
  - **xsd:fractionDigits** : nombre de chiffres après la virgule

# Restriction de type simple

- Restriction de la valeur d'élément age entre 0 et 100

```
<xsd:element name="age">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:integer">  
      <xsd:minInclusive value="0"/>  
      <xsd:maxInclusive value="100"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

# Restriction de type simple

- Restriction sur l'ensemble des valeurs

```
<xsd:element name="car">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Audi"/>
      <xsd:enumeration value="Golf"/>
      <xsd:enumeration value="BMW"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Restriction de type simple

- Restriction sur la série des valeurs

```
<xsd:element name="letter">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:pattern value="[a-z]"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

# Restriction de type simple

- Restriction sur la longueur

```
<xsd:element name="password">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:length value="8"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

# Restriction de type simple

- Restriction sur la longueur

```
<xsd:elementname="password">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:minLength value="5"/>  
      <xsd:maxLength value="8"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```



# liste et union de type simple

```
<xsd:simpleType>  
  <xsd:list itemType="xsd:int"/>  
</xsd:simpleType>
```

```
<xsd:simpleType>  
  <xsd:union memberTypes="xsd:int xsd:boolean"/>  
</xsd:simpleType>
```

# Élément Complexe

- L'élément complexe est un élément XML qui contient les autres éléments et/ou attributs
- Il y a 4 types d'éléments complexes:
  - Élément vide avec au moins un attribut
  - Élément qui contient que des textes avec au moins un attribut
  - Élément qui contient seulement des autres éléments
  - Élément qui contient des autres éléments et des textes

# Élément Complexe

- L'élément complexe personne qui est vide
  - `<personne id="1345"/>`
- L'élément complexe aliment qui contient que des textes
  - `<aliment type="dessert">crème glacée</aliment>`
- L'élément complexe employee qui contient seulement des autres éléments
  - `<employee>`
    - `<firstname>Moussa</firstname> <lastname>Bâ</lastname>`
  - `</employee>`
- L'élément complexe description qui contient des autres éléments et des textes
  - `<description>`
    - `Nous sommes le <date lang="anglais">03.03.99</date> ..`
  - `</description>`

# Lier le Schéma avec le fichier

- On fait référence au schéma dans le document XML en utilisant l'attribut **xsi:noNamespaceSchemaLocation**
- Par exemple:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<person xmlns:xsi=http://www.w3.org/2001/XMLSchema-  
instance xsi:noNamespaceSchemaLocation="non_fic.xsd">  
  
.....  
</person>
```

# Élément Complexe vide

```
<personne id="2505" />
```

```
<xsd:element name="personne">  
  <xsd:complexType>  
    <xsd:attribute name="id" type="xsd:positiveInteger"/>  
  </xsd:complexType>  
</xsd:element>
```

# Élément Complexe avec des éléments

```
<employee>
  <firstname>Moussa</firstname>
  <lastname>Bâ</lastname>
</employee>
```

```
<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

# Élément Complexe avec des éléments

```
<employee>  
  <firstname>Moussa</firstname>  
  <lastname>Bâ</lastname>  
</employee>
```

```
<xsd:element name="employee" type="personinfo" />  
<xsd:complexType name="personinfo">  
  <xsd:sequence>  
    <xsd:element name="firstname"  
type="xsd:string"/>  
    <xsd:element name="lastname" type="xsd:string"/>  
  </xsd:sequence>  
</xsd:complexType>
```

# Élément Complexe avec des éléments et attributs

```
<xsd:element name="auteur">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nom" type="xsd:string"/>
      <xsd:element name="prenom" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:token"/>
  </xsd:complexType>
</xsd:element>
```



## Élément Complexe avec un contenu mixte

```
<xsd:complexType name="personType" mixed="true">  
  <xsd:sequence>  
    <xsd:element name="nom" type="xsd:string"/>  
    <xsd:element name="prenom" type="xsd:string"/>  
  </xsd:sequence>  
</xsd:complexType>  
<xsd:element name="employe" type="personType"/>
```

# Élément Complexe avec contenu simple et attributs

```
<xsd:element name="auteur">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribut name="nom" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

# Indicateur de types complexes

- Indicateurs de l'ordre
  - Tous (All)
  - Choix (Choice)
  - Séquence (Sequence)
- Indicateurs d'occurrence
  - maxOccurs
  - minOccurs

# Indicateur All

- Spécifie que des éléments fils peuvent apparaître en n'importe quel ordre et chaque élément fils doit se produire seulement une fois

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

# Indicateur Choice

- Spécifie qu'on ne peut avoir qu'un élément fils parmi les éléments fils qui peuvent être produits

```
<xsd:element name="person">  
  <xsd:complexType>  
    <xsd:choice>  
      <xsd:element name="employee" type="employee"/>  
      <xsd:element name="member" type="member"/>  
    </xsd:choice>  
  </xsd:complexType>  
</xsd:element>
```

# Indicateur Sequence

- Spécifie que les éléments fils doivent être produits en ordre

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"/>
      <xsd:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

# Indicateur maxOccurs

- Spécifie le nombre maximum de fois qu'un élément peut être produit

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="full_name" type="xsd:string"/>
      <xsd:element name="child_name" type="xsd:string"
maxOccurs="10"/>
```

# Indicateur minOccurs

- Spécifie le nombre minimum de fois qu'un élément peut être produit

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="full_name" type="xsd:string"/>
      <xsd:element name="child_name" type="xsd:string"
        maxOccurs="10" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



# L'extraction de fragments : XPath, XQuery

# XPath

- Comme son nom l'indique, XPath est une spécification fondée sur l'utilisation de chemin d'accès permettant de se déplacer au sein du document XML.
- XPath est un langage avec une syntaxe non XML, permettant d'adresser les différents noeuds ou groupes de noeuds particuliers d'un document XML.

# XPath

- XPath permet d'effectuer des requêtes dans un document XML.
- XPath voit le document XML comme un arbre de noeuds, qu'il permet de parcourir selon des axes (**fils**, **parent**, **ancêtre**, **descendant**, ...) et en sélectionnant les noeuds par leur nom.
- XPath est utilisé par XSLT pour faire de la transformation de documents XML.

# Expressions XPath

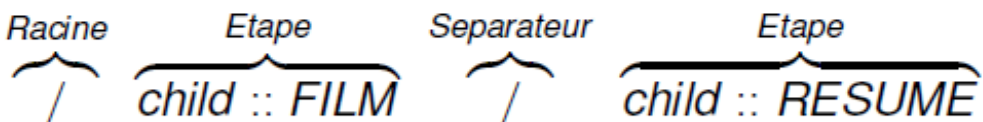
Une **expression XPath** est une séquence d'**étapes** :

$$[/] \text{étape}_1 / \text{étape}_2 / \dots / \text{étape}_n$$

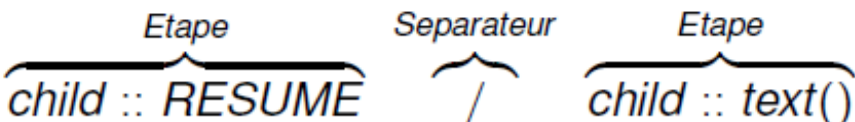
Une **étape XPath** est composé d'un axe, d'un filtre et d'un prédicat (optionnel) :

$$\text{axe}::\text{filtre}[\text{predicat}]$$

Une expression peut être

- **absolu** : 

Le nœud contexte est la racine du document.

- **relatif** : 

Le nœud contexte est un nœud dans le document (pas forcément la racine).

# Axes

- **child** : contient les enfants directs du noeud contextuel.
- **descendant** : contient les descendants du nœud contextuel. Un descendant peut être un enfant, un petit enfant...
- **parent** : contient le parent du noeud contextuel, s'il y en a un.
- **ancestor** : contient les ancêtres du noeud contextuel. Cela comprend son père, le père de son père... Cet axe contient toujours le noeud racine, excepté dans le cas où le nœud contextuel serait lui-même le noeud racine.

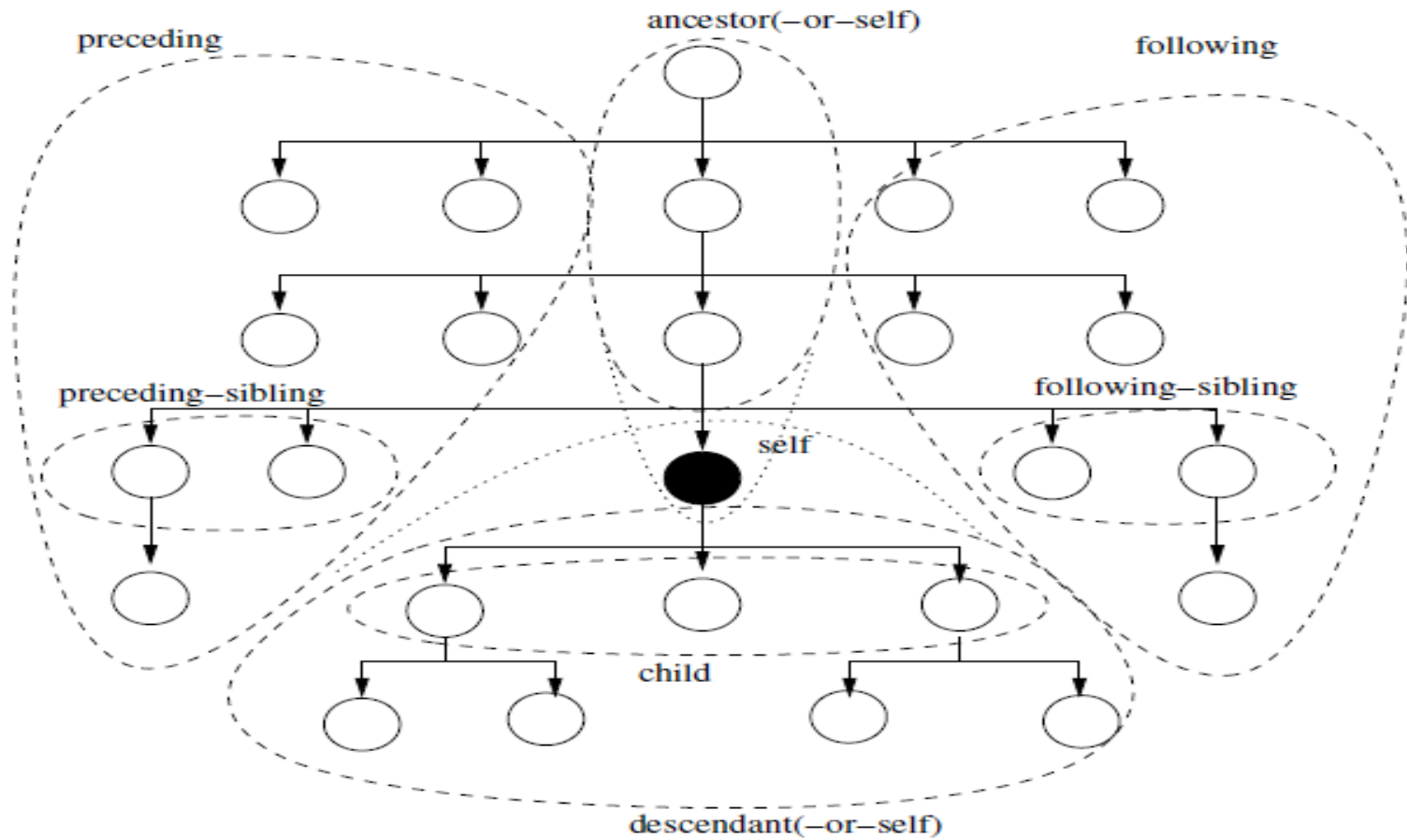
# Axes

- **following-sibling** : contient tous les noeuds frères qui suivent le noeud contextuel. Dans le cas où ce noeud est un attribut ou un espace de noms, la cible suivante est vide.
- **preceding-sibling** : contient tous les frères prédécesseurs du noeud contextuel ; si le noeud contextuel est un attribut ou un espace de noms, la cible précédente est vide.
- **following** : contient tous les noeuds du même document que le noeud contextuel qui sont après le noeud contextuel dans l'ordre du document, à l'exclusion de tout descendant, des attributs et des espaces de noms.
- **preceding** : contient tous les prédécesseurs du noeud contextuel ; si le noeud contextuel est un attribut ou un espace de noms, la cible précédente est vide.

# Axes

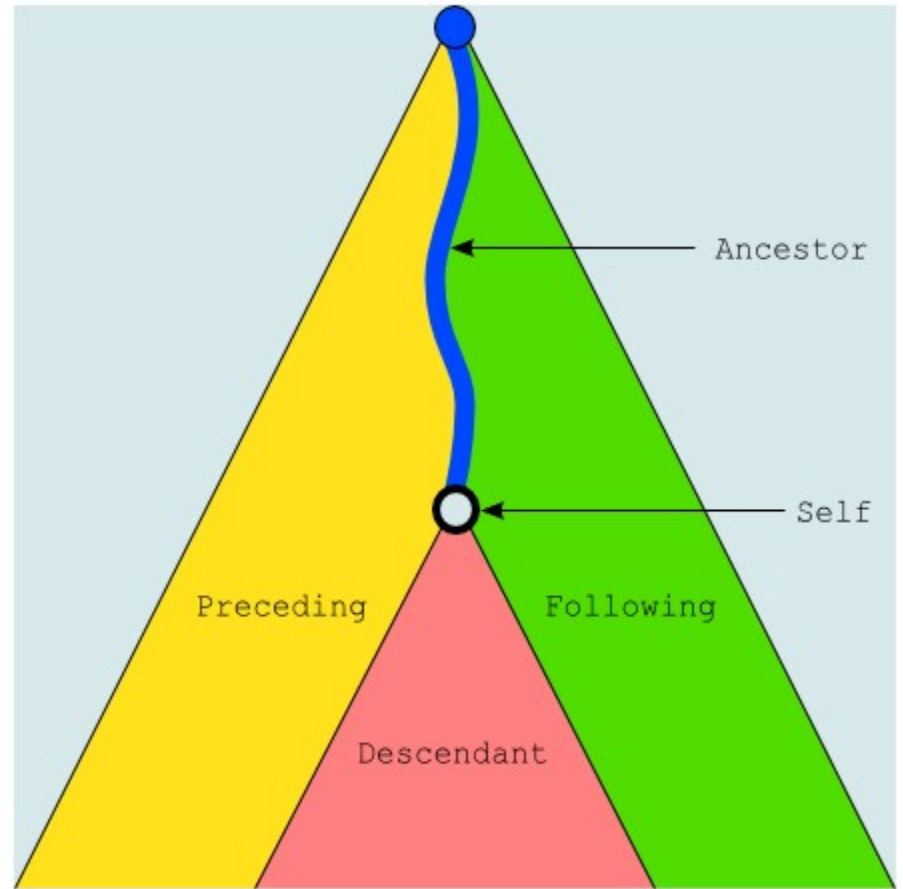
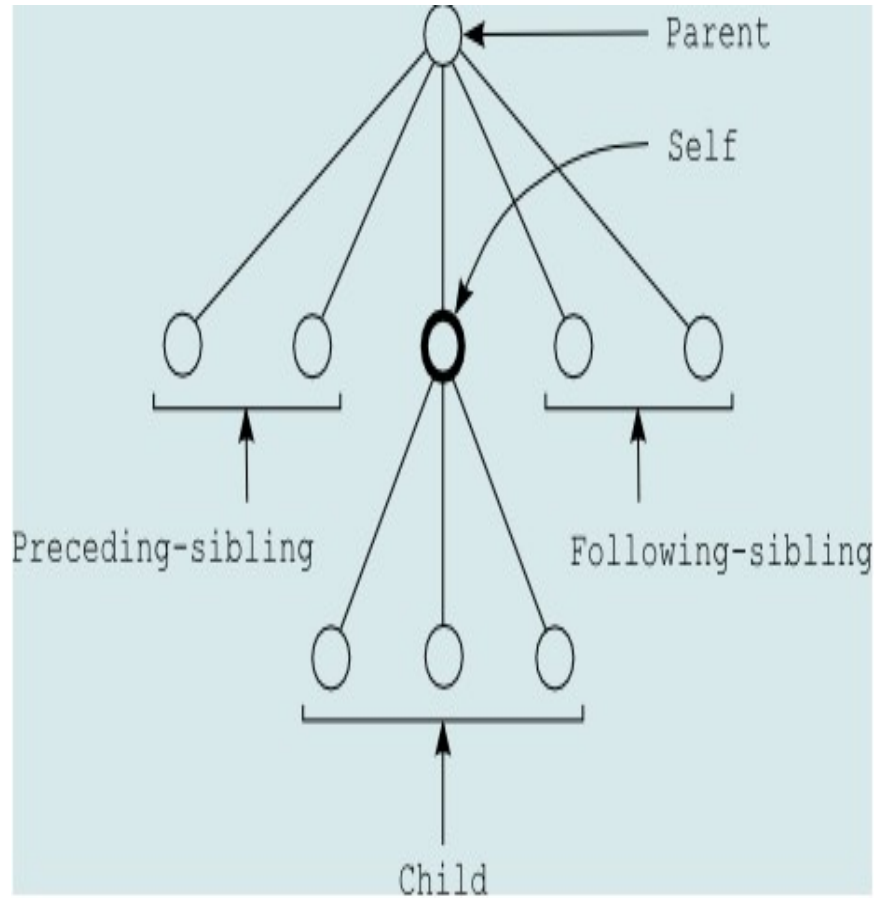
- **attribute** : contient les attributs du noeud contextuel ; l'axe est vide quand le noeud n'est pas un élément.
- **namespace** : contient tous les noeuds des espaces de noms du noeud contextuel ; l'axe est vide quand le noeud contextuel n'est pas un élément.
- **self** : contient seulement le noeud contextuel.
- **descendant-or-self** : contient le noeud contextuel et ses descendants.
- **ancestor-or-self** : contient le noeud contextuel et ses ancêtres. Cet axe contiendra toujours le noeud racine

# Axes





# Axes



# Filtres XPath

- **Filtrage par le nom** : pour les types de nœuds qui ont un nom : **Element**, **ProcessingInstruction** et **Attr**.
- **Filtrage par le type DOM** :
  - \* : nœuds de type Élément ou Attribut
  - **text()** : nœuds de type Text
  - **comment()** : nœuds de type Comment
  - **processing-instruction()** : nœuds de type ProcessingInstruction
  - **node()** recouvre tous les types de noeud

# Prédicats XPath

- **Prédicat** : expression booléenne constituée d'un ou plusieurs tests, composés avec les connecteurs logiques habituels **and** et **or**.
- **Test** :
  - une expression XPath qui est convertie en booléen (règles de conversion implicites).
  - une comparaison ou un appel de fonction.
- **Exemples** :
  - [attribute::A='1']
  - [attribute::A=attribute::B]
  - [position()=1], [position()=last()]
  - [child::B]

# Exemples

La position d'un nœud dépend de l'axe choisi :

- **child::\*[3]** : le 3e enfant
- **child::\*[position()=3]** : idem
- **child::\*[last()]** : le dernier enfant
- **descendant::\*[last()]** : le dernier descendant
- **ancestor::\*[1]** : le premier ancêtre du noeud contextuel.
- **preceding-sibling::\*[last()]** : le dernier frère qui précède le nœud contextuel.

# Syntaxe abrégée

- Axe par défaut : **/A** correspond à **/child::A**
- Étape par défaut : **A//B** exprime **A/descendant-or-self::node()/B**
- L'expression **“..”** est équivalente à **parent::node()**.
- L'expression **“.”** est équivalent à **self::node()** et désigne le nœud courant.
- **child::personne/child::email ==> personne/email**
- **/child::carnet/descendant::email==> /carnet//email**

# Fonctions sur les nœuds

- **last()** : nombre de nœuds inclus dans le contexte courant
- **position()** : retourne le numéro d'ordre du nœud courant
- **count(liste de nœuds)**
- **name(liste de nœuds)**

# Exemples

- **child::chapter/child::section** ou **chapter/section**
  - élément section fils d'un nœud chapter lui-même fils d'un nœud du contexte.
- **parent::node()/child::chapter** ou **../chapter**
  - élément chapter frère d'un nœud du contexte.
- **self::node()/descendant-or-self::section** ou **../section**
  - élément section descendant d'un nœud du contexte.

# Exemples

- **child::section[position()=2]** ou **section[2]**
  - deuxième élément section fils du nœud courant.
- **descendant::p/following-sibling::em[position()=1]**  
ou **./p/following-sibling::em[1]**
  - premier frère em d'un élément p descendant du nœud courant.
- **child::section[child::title]** ou **section[title]**
  - élément section fils du nœud courant et ayant un fils title.



# Exemples

- **child::section[attribute::title]** ou **section[@title]**
  - élément section fils du nœud courant et ayant un attribut title.
- **child::section[attribute::title='Un titre']** ou **section[@title='Un titre']**
  - élément section fils du nœud courant et ayant un attribut title égal à la chaîne Un titre.

# Exemples

- **chapter[count(child::section) > 1]**
  - élément chapter ayant plus d'un élément section comme fils.
- **//@\*[name() != 'id']**
  - attributs autres que l'attribut id.
- **section[2][@type='dual']**
  - deuxième élément section du nœud courant si son attribut type vaut dual.

# XPath : Résumé

- XPath est un langage pour extraire des nœuds dans un arbre XML :
  - On navigue dans l'arbre grâce à des axes.
  - Un chemin de localisation est une séquence d'étapes.
  - Dans chaque étape on choisit un axe, un filtre et éventuellement des prédicats.
  - Le résultat d'une étape (d'une séquence d'étapes) est une séquence de nœuds.

# XQuery

- XQuery est un langage de "programmation" puissant pour extraire des données XML
- Type de données: un seul "document" ou encore des collections sous forme de:
  - fichiers
  - bases de données XML
  - XML "en mémoire" (abres DOM)

# XQuery

- Ex: Une requête qui permet très simplement de renvoyer tous les noeuds **auteur** du document `biblio.xml`.
  - Aucune mise en forme n'étant pratiquée, les nœuds renvoyés le sont avec leurs balises et leur contenu telqu'il s'exprime dans le document.
  - `doc("biblio.xml")//auteur`

# Les expressions FLWOR

- FLWR = "For-Let-Where-Order-Return"
- rappelle l'idée du select-from-where-..-order-by de SQL

# Document Example

- <bibliothèque>
  - <livre année="2002">
    - <titre> Prélude à fondation </titre>
    - <auteur><nom> Asimov </nom><prénom> Isaac </prénom></auteur>
    - <éditeur> Pocket </éditeur>
    - <prix>6</prix>
  - </livre>
  - <livre année="1995">
    - <titre> La marche des millénaires </titre>
    - <auteur><nom> Asimov </nom><prénom> Isaac </prénom></auteur>
    - <auteur><nom> White </nom><prénom> Frank </prénom></auteur>
    - <éditeur> Flammarion </éditeur>
    - <prix>8.45</prix>
  - </livre>
  - <livre année="2005">
    - <titre> Avant Dune </titre>
    - <auteur><nom> Anderson </nom>
    - <prénom> Kevin J. </prénom></auteur>
    - <auteur><nom> Herbert </nom><prénom> Brian </prénom></auteur>
    - <éditeur> Pocket </éditeur>
    - <prix>12.3</prix>
  - </livre>
- </bibliothèque>

# Les expressions FLWOR

- **for** = itération sur une liste de fragments xml
- **let** = association du résultat d'une expression à une variable
- **where** = condition de sélection
- **order** = tri des résultats
- **return** = expression à retourner



# "for"

- **for \$variable in expression\_recherche  
RETURN**
- "for" associe à chaque \$variable une valeur (fragment XML) trouvé pour l'expression XQuery
- Exemple:
- **for \$t in doc("biblio.xml")//livre**

# let

- "**let**" permet d'assigner une valeur à une variable
- requête qui permet d'affecter la variable **\$b** successivement avec le contenu des noeuds **livre** du document **biblio.xml** (clause **for**) et d'affecter à **\$a** l'ensemble des noeuds **auteur** contenus dans chaque itération de **\$b** (clause **let**)
- **for \$b in doc("biblio.xml")//livre**
- **let \$a := \$b/auteur**
- **return <livre nb\_auteurs="{count(\$a)}"> { \$a } </livre>**

# where

- permet de définir une condition de sélection
- requête pour extraire le **titre** des livres dont le premier noeud fils **auteur** contient comme valeur de **nom** Mansour
- **for \$a in doc("biblio.xml")//livre**
- **where \$a/auteur[1]/nom eq "Mansour"**
- **return \$a/titre**

# order

- permet de trier les résultats
- requête pour extraire le **titre** des livres dont le premier noeud fils **auteur** contient comme valeur de **nom** Mansour
- **for \$a in doc("biblio.xml")//livre**
- **where \$a/auteur[1]/nom eq "Mansour"**  
**Order by \$a/titre**
- **return \$a/titre**

# Expressions conditionnelles

- Structure **if - then - else**
- **if** : teste une condition.
- **then** : opération en cas de réponse positive au test.
- **else** : opération en cas de réponse négative au test.
- Ex : Requête qui va tester pour chaque élément **livre** de biblio.xml si la valeur de son attribut **année** est supérieure à 2000 et renvoyer la valeur 'récent' si c'est la cas et 'ancien' sinon.
- for \$b in doc("biblio.xml")//livre
- return
- if (\$b/@année > 2000)
- then 'récent'
- else 'ancien'

# eXtensible Stylesheet Language / Transformation (XSLT)

# XSLT

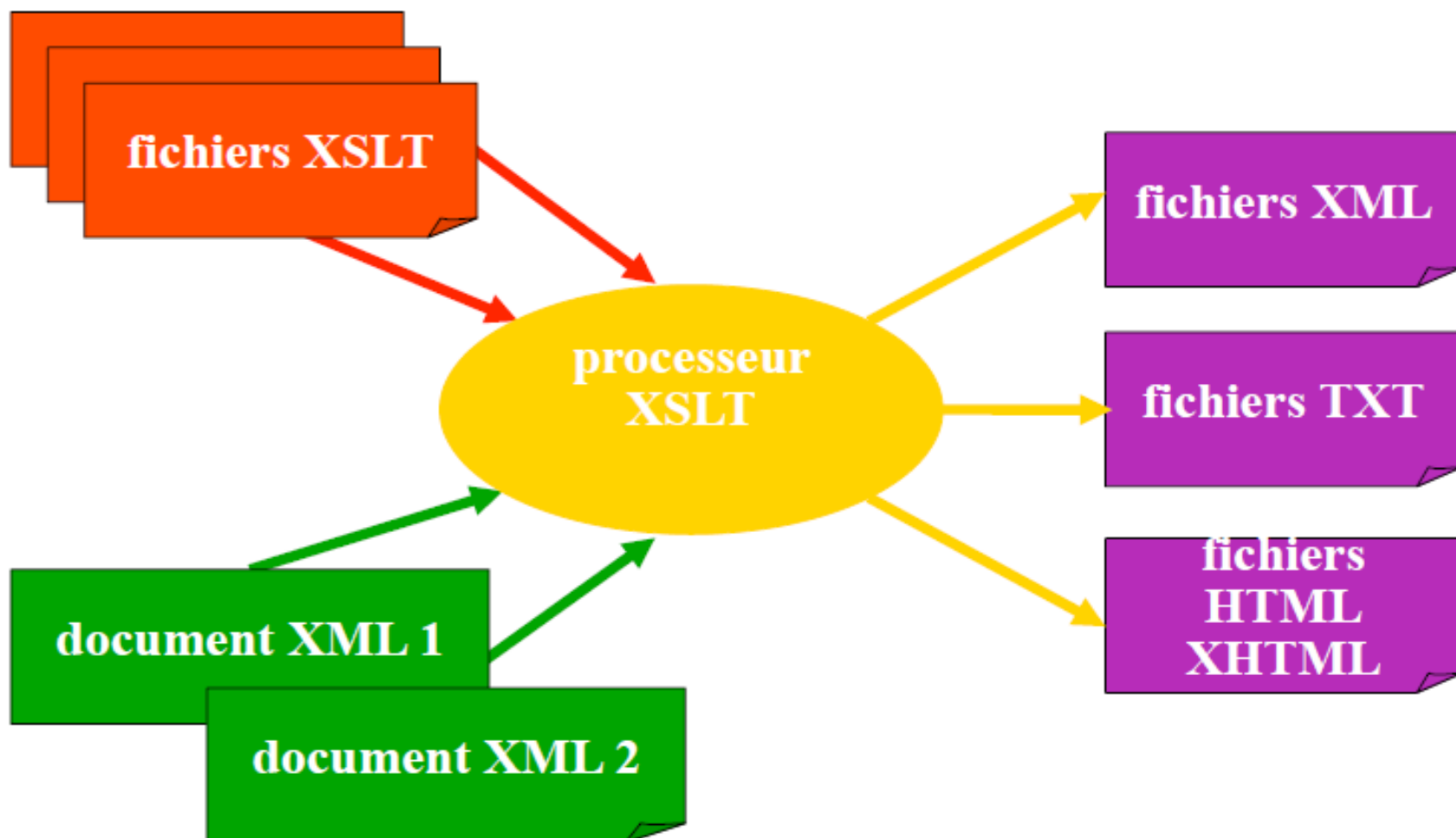
- XSL (eXtensible Stylesheet Language) est un langage de feuilles de styles : plus de fonctionnalités que CSS
  - Filtrage de données XML
  - Présentation des données dans n'importe quel ordre
  - Modification et ajout d'information
  - Peut inclure des scripts
- Prise en compte de XML en plus du HTML
- XSL est une Recommandation du W3C composée de :
  - XSLT: un langage de transformation
  - XPath: déjà étudié, sélection de nœuds
  - XSL-FO: un jeu d'instructions de formatage en XML destiné à la présentation

# XSLT

- XSLT permet de transformer un document XML en un autre document XML ou bien dans un autre format, à des fins de:
  - affichage: on peut obtenir un document HTML pour la visualisation (comparable à CSS). On peut également produire des fichiers textes, PDF, ...
  - adaptation: on veut adapter les données contenues dans le document pour obéir à une DTD donnée ...



# XSLT



# XSLT

- XSLT est un langage de transformation d'arbre. Il s'agit d'une API qui réalise le passage d'un document XML vers un document texte (souvent au format XML lui aussi)
- Puissance d'un langage de programmation (variables, itération, conditionnelle, ...)
- Langage à base de règles (templates)
  - template = sélecteur + forme à générer
  - application des règles basée sur un parcours récursif de l'arbre et des priorités

# XSLT

- Une feuille de style XSLT est un document XML

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

**<xsl:stylesheet version="1.0"**

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

**... instructions ...**

**</ xsl:stylesheet>**

- L'appel XSLT à partir du document XML

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

**<?xml-stylesheet type="text/xsl" href=" monfic.xsl"?>**

# <xsl:template>

- Une feuille de style XSLT est constituée d'une séquence de règles de transformation de la forme :

<xsl:template match="**chemin**">

**... transformations ...**

</xsl:template>

- Une règle comporte deux parties :
  - **un modèle de chemin** exprimé en termes du langage XPath (identifie un ensemble de noeuds)
  - **une transformation** sous la forme d'une séquence de caractères ou d'éléments dont certains sont des instructions XSLT

# <xsl:template>

- Règles par défaut :
  - Pour la racine et les autres types de noeuds, appliquer les patrons (définis ou patrons par défaut) aux noeuds fils

```
<xsl:template match="/ | *">  
  <xsl:apply-templates/>  
</xsl:template>
```
  - Pour les nœuds textes et attributs on recopie leurs valeurs dans le flot de sortie

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

## <xsl:value-of>

- La règle suivante se déclenche dès qu'une balise <project> est trouvée
- Elle insère dans le document de sortie le contenu de l'élément <title> qui se trouve à l'intérieur d'un sous-élément <problem>

**<xsl:template match="project">**

**<P> <xsl:value-of select="problem/title"/> </P>**

**</xsl:template>**

# <xsl:copy-of>

- <xsl:copy-of select="path"/>
  - Recopie dans le flot de sortie le fragment du document à transformer sélectionné par la requête XPath à partir du nœud contexte
  - Ce fragment peut être constitué de plusieurs nœuds
- Exemple :
  - <xsl:copy-of select="gdt/expose"/>
  - Recopie les éléments **expose** désignés et leur contenu

# <xsl:element>

- `<xsl:element name="balise">... </xsl:element>`
  - construit un nœud élément de nom balise avec les attributs et contenus définis dans le corps de l'élément `xsl:element`
- Exemple :
  - `<xsl:element name="p"><xsl:value-of select="para" /></xsl:element>`
  - Crée une balise p contenant le texte présent dans l'élément `para` du document d'origine



# <xsl:attribute>

- `<xsl:attribute name="att">valeur</xsl:attribute>`
  - construit un attribut nommé “att” (à utiliser à l’intérieur d’un élément `xsl:element`)
- Exemple :

```
<xsl:element name="image">  
  <xsl:attribute name="src">  
    <xsl:value-of select="@adresse"/>  
  </xsl:attribute>  
  ....  
</xsl:element>
```

  - Crée une balise image, ayant un attribut src dont la valeur est celle de l’attribut adresse de l’élément d’origine

# Syntaxe courte

- Dans une feuille de style, on peut en fait insérer directement de nouvelles balises :
  - `<image> ... </image>`
- Pour faire référence au document d'origine, un élément peut contenir dans ses attributs des « expressions attributs » de la forme `{path}`, où `path` est une expression XPath qui est évaluée vers une chaîne de caractères :
  - `<image src="{@adresse}"></image>`

# xsl:apply-templates

- Cette balise permet de déclencher l'application de règles sur un ensemble de nœuds
- `<xsl:apply-templates/>`
  - La liste des nœuds à traiter est constituée des nœuds fils du nœud contexte
- `<xsl:apply-templates select="path"/>`
  - La liste des nœuds à traiter est constituée des nœuds atteints par le chemin `path` depuis le nœud contexte

# <xsl:for-each>

**<xsl:for-each select="cheminXPath">**

**... instructions ...**

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

- applique les instructions à tous les nœuds désignés par le chemin XPath.
- Un moyen simple pour remplir des tableaux html avec des contenus d'élément XML

# <xsl:choose>

- Combiné avec **<xsl:when>** et **<xsl:otherwise>**, permet de construire des tests conditionnels à l'instar du switch de Java.

**<xsl:choose>**

**<xsl:when test="condition">**

**instructions**

**...**

**</xsl:when>**

**...**

**<xsl:otherwise>**

**instructions**

**...**

**</xsl:otherwise>**

**</xsl:choose>**

## <xsl:if>

- On peut tester la valeur (booléenne) d'une expression XPath et exécuter un template si la valeur est vraie : (en pratique on teste souvent l'existence d'un noeud)
- **<xsl:if test="expression"> ... </xsl:if>**
- Exemple :
- **<xsl:if test="@langue='français'">Ce livre est en français.</xsl:if>**

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="test.xsl"?>
<catalog>
  <cd>
    <title>Empire</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hideyourheart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</compnay>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>

```



Title	Artist
Empire	Bob Dylan
Hide your heart	Bonnie Tyler

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
    <body>
      <h2>My CD ollection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th>Title</th> <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td>
              <xsl:value-of select="./title"/>
            </td>
            <td>
              <xsl:value-of select="./artist"/>
            </td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

```

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="test.xsl"?>
<catalog>
  <cd>
    <title>Empire</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hideyourheart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</compnay>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>

```



Title: **Empire**  
Artist: **Bob Dylan**

Title: **Hideyourheart**  
Artist: **Bonnie Tyler**

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="cd">
    <p>
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="artist"/>
    </p>
  </xsl:template>
  <xsl:template match="title">
    Title:
    <span style="color:#ff0000">
      <xsl:value-of select="."/>
    </span>
    <br/>
  </xsl:template>
  <xsl:template match="artist">
    Artist:
    <span style="color:#00ff00">
      <xsl:value-of select="."/>
    </span>
    <br/>
  </xsl:template>
</xsl:stylesheet>

```