

Université Mohammed Premier

Faculté des Sciences Département d'Informatique Oujda

Technologie Web Avancée

XML : eXtensible Markup Language

SMI S6

2019/2020

Pr M.SERRHINI

Le HTML

- le standard du développement Web.
- Il est amené à disparaître progressivement, étant remplacé par le XHTML, un langage qui lui est extrêmement similaire, mais permettant la production de documents aux normes XML.

Inconvénients du HTML

- Son champ d'action limité : il n'est ainsi pas possible de définir *autre chose* qu'une page Web.
- On ne peut par exemple pas ajouter de nouveaux éléments (on pourrait imaginer insérer des équations mathématiques, mais ce n'est pas possible).

Document HTML: les informations sur un livre

```
<html>
<body>
<ul>
<li> XML: le guide de l'utilisateur </li>
<li> E.R.Harold </li>
<li> Eyrolles </li>
<li> 1999 </li>
</ul>
</body>
</html>
```

Dans un fichier nommé teste.xml, on présente les informations sur le livre de cette manière :

- **Document XML**

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

```
<livre>
```

```
<titre> XML: le guide de l'utilisateur </titre>
```

```
<auteur> E.R.Harold </auteur>
```

```
<edition> Eyrolles </edition>
```

```
<annee> 1999 </annee>
```

```
</livre>
```

- Lisibilité: données auto-documentées
- Séparation entre l'information et la présentation
- Traitement par d'autres applications

Affiché dans le navigateur, le fichier teste.xml donne ce résultat



```
- <livre>
  <titre> XML: le guide de l'utilisateur </titre>
  <auteur> E.R. Harold </auteur>
  <edition> Epyrolles </edition>
  <annee> 1999 </annee>
</livre>
```



- XML: le guide de l'utilisateur
- E.R. Harold
- Epyrolles
- 1999



- Le XML est un langage de balises [Markup Language].
- HTML possède un jeu de balises fixes.
- Le langage est basé sur le concept de balisage des données
- XML n'a pas de balises prédéfinies mais permet de se définir ses propres balises.
- HTML mélange le contenu et la présentation
(améliorations possibles cependant avec CSS).
 - XML ne décrit que du contenu pur.
- Pour la présentation on pourra utiliser du XSL ou du CSS.

XML

- Le langage est basé sur le concept de balisage des données
- Idéal pour l'échange de données semi-structurées
- Utilisable entre machines
- XML, c'est don un méta-langage universel pour structurer les données qui permet aux utilisateurs de délivrer du contenu...

depuis les applications à d'autres applications browsers par exemple

- XML promet de standardiser la manière dont l'information est:
 - échangée (XML)
 - personnalisée/présentée (XSL/CSS)
 - recherchée (XPath/XSLT/XQuery)
- • sécurisée (Encryption, Signature)
- • liée (XLink)
-

XML

- Le XML est un dérivé du SGML. Il tente de se servir des principes de simplicité du HTML et de la souplesse SGML.

Le plus important point commun avec le SGML est le fait que tout document XML *peut* être basé sur une DTD ou un Schéma.

- Cette association n'est cependant pas obligatoire,
- Un fichier XML peut très bien se suffire à lui même.

Que signifie XML ?

- eXtensible: une infinité de balises;
- Markup: identifier les # éléments d'un document au moyen de balises;
- Language: respecter un certain nombre de règles.

Pourquoi XML ?

Définir vos propres langages d'échange

- commande, facture, bordereau de livraison, etc.
- Modéliser des données et des messages
- *Document Type Definition (DTD)*
- types et éléments agrégés (*XML Schema Definition*)
- passerelle avec *Unified Modelling Language (UML)*
- Publier des informations
- indépendante du format
- mise en forme avec CSS et XSL
- présentation possible en XHTML, PDF, WML,... Archiver des données
 - auto-description des archives

• **Avantage de XML**

- Dans un document XML, la mise en forme des données est totalement séparée des données elles-mêmes.
- Cela permet de séparer complètement l'information (le contenu) de son apparence (le contenant)
- donc de fournir plusieurs types de sortie pour un même fichier de données, en fonction de l'utilisateur ou de l'application demandeuse (tableau, graphique, image, animation multimédia, fichier HTML, fichier PDF...).

Avantages de XML

- la possibilité de créer les éléments que l'on désire permet de rendre le fichier lui-même lisible - et modifiable - par un être humain : on peut donner aux informations contenues dans un tel fichier le nom que l'on veut, et les ordonner selon son désir.
 - Un document XML peut ainsi prévoir plusieurs cibles: l'écran d'un téléphone portable, celui d'un ordinateur de bureau, une base de données, une application logicielle, etc.
- Il est également possible d'effectuer des sélections par tri, des générations automatiques de tables des matières et bien d'autres fonctions encore, grâce au langage de feuilles de style XSL.

Document XML

- Un document XML peut être associé à:
- Un DTD ou un schéma pour décrire la structure du document XML. Le DTD ou/et schéma permettent de définir son propre langage basé sur XML
- Une feuille de style pour présenter les données
- vocabulaire (balises)
- grammaire (imbrications)
- Deux types de documents
- *well-formed document*
- *valid document*

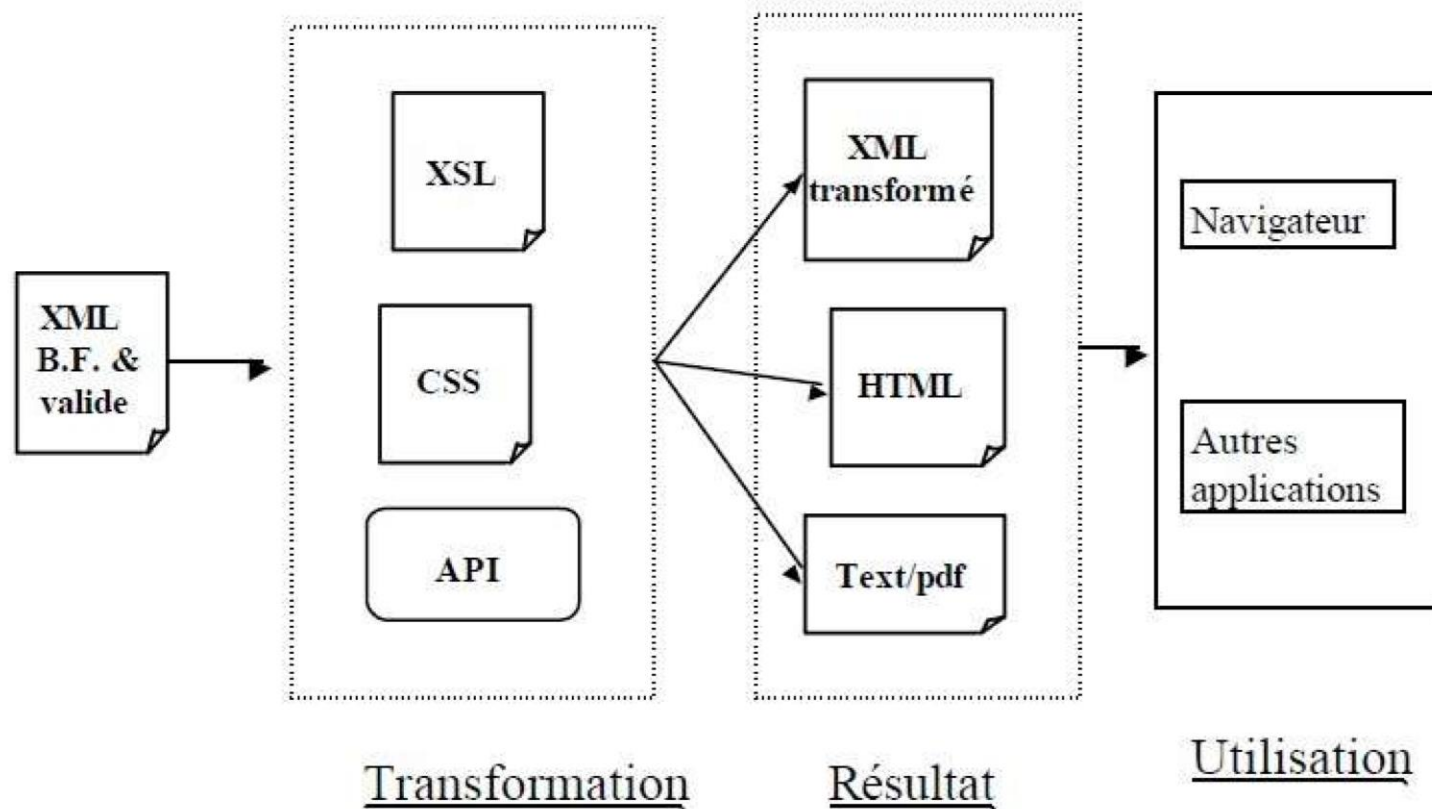
Document bien formé (*well-formed*)

- Commence par une déclaration XML (attribut version obligatoire) avec possibilité de choisir un encodage (le défaut est utf-8): `<?xml version="1.0" encoding="ISO-8859-1"?>`
- Structure hiérarchique:
- les balises d'ouverture et de fermeture doivent apparaître et correspondre
- pas de croisements de type `<i>......</i> `
- sensible à la casse: "LI" n'est pas égal à "li" par exemple
- balises "EMPTY" utilisent la syntaxe XML "auto-fermante": `
`
- les valeurs d'attributs sont quotés: ``
- un seul élément racine (root): l'élément root ne peut apparaître qu'une fois et ne doit pas apparaître dans un autre élément
- Caractères spéciaux (!!): `<`, `&`, `>`, `"`, `'` utilisez `<`, `&`, `>`, `"`, `'` à la place dans un texte !
- les espaces sont préservés
-

Document valide (*valid*)

- Un document “valide” doit être:
- “well-formed” (formé correctement)
- être associé à une DTD (ou une autre grammaire)
- et être conforme à cette DTD ou une grammaire d’un autre type comme XSD (*XML Schema Definition*) ou RelaxNG

Cycle de vie d'un document XML



Générer un document HTML à partir de XML

Pour produire un fichier HTML à partir de données mises sous format XML, il faut :

- Créer éventuellement un fichier définissant les balises utilisables ;
- Créer le fichier de données XML ;
- Créer la feuille de style XSL permettant la production du fichier HTML ;
- Créer éventuellement une feuille de style CSS.

Structure des documents XML

- **Prologue**: Une déclaration d'un document XML
- **Instructions de traitements**: fournissent de l'information à une application XML
- Définition optionnelle de **type de document** (DTD)
- Références aux **entités**
- Différents éléments du document
- **Commentaires** (facultatif)
- N.B: un document XML a l'extension **.xml**
-
-

- **Exemple : une bibliographie**

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

```
<biblio>
```

```
<livre>
```

```
<titre> Les Misérables</titre>
```

```
<auteur> Victor Hugo</auteur>
```

```
<nb_tomes> 3 </nb_tomes>
```

```
</livre>
```

```
<livre>
```

```
<titre> L'Assomoir</titre>
```

```
<auteur> Émile Zola </auteur>
```

```
</livre>
```

```
<livre lang="en">
```

```
<titre> David Copperfield </titre>
```

```
<auteur> Charles Dickens </auteur>
```

```
<nb_tomes> 3 </nb_tomes>
```

```
</livre>
```

```
</biblio>
```

Structure d'un document XML

- un fichier XML est composé d'un prologue, d'un élément racine et d'un arbre.

Structure d'un document XML

- Cet arbre est constitué d'éléments imbriqués les uns dans les autres (ayant une relation parentenfant) et d'éléments adjacents.

Structure d'un document XML

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

Prologue

<!DOCTYPE parc-machine SYSTEM "comp.dtd">

Type de doc.

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

II

<!-- ce document decrit le parc-machine -->

Commentaire

<parc-machine>

<machine idf="mach001">

<modele>HP-PC</modele>

<service>23/02/2000</service>

<ip>212.217.30.100</ip>

<systeme>

<nom>Windows</nom>

<version>2000Profesional</version>

</systeme>

<ram unite="Mo">256 </ram>

<disque unite="G0">20</disque>

<processeur>Intel PentiumIII</processeur>

<horloge>700Mhz</horloge>

<administrateur src="photo.jpg"> F. H.</administrateur>

</machine>

<machine>

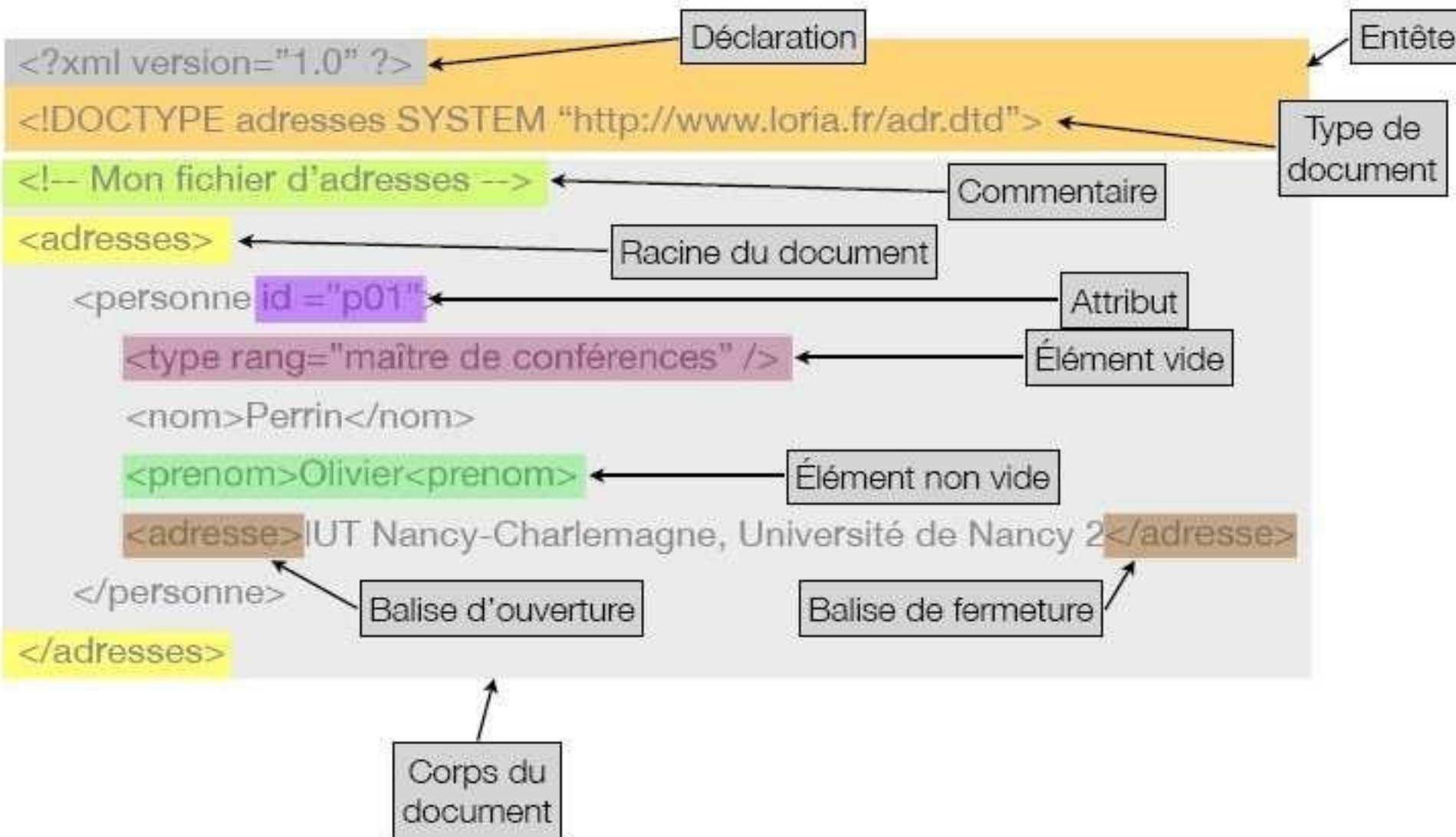
.....

</machine>

</parc-machine>

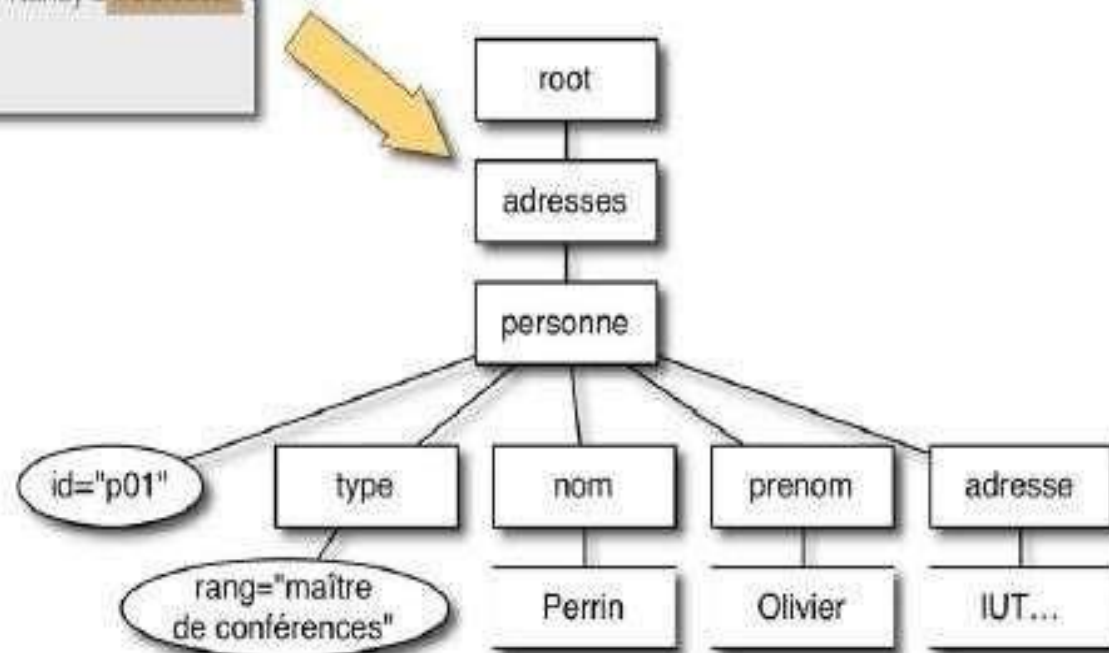
Les éléments

Structure d'un document XML



- C'est un arbre

```
<?xml version="1.0" ?>
<!DOCTYPE adresses SYSTEM "http://www.loria.fr/adr.dtd">
<!-- Mon fichier d'adresses -->
<adresses>
  <personne id="p01">
    <type rang="maître de conférences" />
    <nom>Perrin</nom>
    <prenom>Olivier</prenom>
    <adresse>IUT Nancy-Charlemagne, Université de Nancy 2</adresse>
  </personne>
</adresses>
```



1. le prologue-généralités

- le prologue, constitué de la déclaration XML, et éventuellement d'une déclaration de type de document (une DTD);
- L'élément biblio est notre élément racine (en anglais : document element); il est constitué de trois éléments livre. Dans chacun d'entre eux nous retrouvons la même composition, c'est-à-dire :
 - un élément titre,
 - un élément auteur et éventuellement
 - un élément nb tomes.
- L'élément livre, de plus, a un attribut lang;

2.Le prologue :Déclaration XML

- Cette déclaration fait partie des "instructions de traitement". Exemple **<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>**
- **version** : version du XML utilisée dans le document, 1.0 en ce qui nous concerne
- **encoding** : le jeu de codage de caractères utilisé. Le jeu de caractère standard pour la France est le *ISO-8859-1*
- **standalone** : dépendance du document par rapport à une déclaration de type de document. Si standalone a la valeur yes, le processeur de l'application n'attend aucune déclaration de type de document extérieure au document. Sinon, le processeur attend une référence de déclaration de type de document. La valeur par défaut est no.

Cette déclaration est facultative, mais il est préférable de l'utiliser, auquel cas les attributs version, encoding et standalone doivent être placés *dans cet ordre*. Si elle est utilisée, elle doit être placée en toute *première ligne* du document XML.

2.Le prologue : Instructions de traitement

- Une instruction de traitement est une instruction interprétée par l'application servant à traiter le document XML.
- Elle ne fait pas totalement partie du document. Les instructions de traitement qui servent le plus souvent sont la déclaration XML ainsi que la déclaration de feuille de style. Exemple d'instruction de traitement :

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

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

2. Le prologue : Déclaration de type de document (DTD)

- Cette déclaration, lorsqu'elle est présente, permet de définir la structure du document. Elle peut être de deux types, externe ou interne.

Exemple de déclaration de type de document :

- **<!DOCTYPE biblio SYSTEM biblio.dtd">**

3. Les commentaires

- En XML, les commentaires se déclarent de la même façon qu'en HTML. Ils commencent donc par `<!--` et se terminent par `-->`.
- Ils peuvent être placés à n'importe quel endroit tant qu'ils se trouvent à l'*extérieur* d'une autre balise.

4. L'arbre d'éléments

- Un document XML peut se représenter sous la forme d'une arborescence d'*éléments*. Cette arborescence comporte une racine (unique), des branches et des feuilles.

Élément racine

L'élément-racine (en anglais : document element) est, comme son nom l'indique, la base du document XML.

Il est unique et englobe *tous* les autres éléments. Il s'ouvre juste après le prologue, et se ferme à la toute fin du document. Dans l'exemple ci-dessus, l'élément racine est biblio.

Les éléments

Les éléments forment la structure même du document : ce sont les branches et les feuilles de l'arborescence. Ils peuvent contenir du texte, ou bien d'autres éléments, qui sont alors appelés "éléments enfants", l'élément contenant étant quant à lui appelé logiquement "élément parent".

- Exemple d'élément contenant du texte :
<titre>Les Misérables</titre>
- Exemple d'élément contenant d'autres éléments :
<livre>
 <titre>L'Assomoir</titre>
 <auteur>Émile Zola</auteur> </livre>

Les attributs

- Tous les éléments peuvent contenir un ou plusieurs attributs. Chaque élément ne peut contenir qu'une fois le même attribut. Un attribut est composé d'un nom et d'une valeur. Il ne peut être présent que dans la balise *ouvrante* de l'élément (par exemple, on n'a pas le droit d'écrire `</livre lang="en">`).
- Exemple d'utilisation d'un élément avec attribut :
`<instrument type="vent">trompette</instrument>`
- Exemple d'utilisation d'un élément vide avec attributs :
``

Les entités

- Il existe deux sortes d'entités, définissables et définies. Elles peuvent être analysables ou non, internes ou externes. La déclaration des entités s'effectue au sein de la DTD. Elles peuvent être utilisées aussi bien dans la DTD que dans le document XML.

Les entités

Certains caractères ayant un sens précis en XML, il est nécessaire de leur trouver un remplaçant lorsque l'on a besoin de les insérer dans un document. On a recours dans ce cas à des entités prédéfinies.

Ces entités sont :

Caractère	Entité
&	&
<	<
>	>
"	"
'	'

Table 1: Liste des entités prédéfinies

Exemple utilisation

- 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. Les données constituent le cœur du document
- Comme certains caractères sont réservés à la syntaxe XML, il faut être vigilant lors de l'écriture des données.

Exemple :

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

</calcul>

- Voici la liste des entités prédéfinies :
- < équivalent de < (*less than*) ;
- > équivalent de > (*greater than*) ;
- & équivalent de & (*ampersand*) ;
- " équivalent de " (*quote*) ;
- • ' équivalent de ' (*apostrophe*).

L'exemple précédent peut donc être correctement réécrit :

If (a<b et b>c)

Les sections CDATA

- Une section CDATA est une section pouvant contenir toute sorte de chaîne de caractères. Une section CDATA permet de définir un bloc de caractères ne devant pas être analysés par le processeur XML.
- Ceci permet entre autres de garder dans un bloc de texte un exemple de code à afficher tel quel.
- Exemple d'utilisation de CDATA :
`<![CDATA[Une balise commence par un < et se termine par un >.]>`

5. Règles de composition

Un certain nombre de règles de base doivent être respectées :

- Un nom d'élément ne peut commencer par un chiffre. Si le nom n'est composé que d'un seul caractère, ce doit être une lettre comprise entre "a" et "z" pour les minuscules, "A" et "Z" pour les majuscules.
- S'il est composé d'au moins deux caractères, le premier peut être "_" ou ":".
- Le nom peut ensuite être composé de lettres, chiffres, tirets, tirets bas et deux points.
- La syntaxe XML est sensible à la casse (le format distingue majuscules et minuscules).

5. Règles de composition

- Toutes les balises portant un contenu non vide doivent être fermées.
- La balise de début, la balise de terminaison et le contenu entre deux sont globalement appelés *élément* ;
- Les balises n'ayant pas de contenu doivent se terminer par `/>` (voir la balise `` ci-dessus) ;
- Les noms d'attributs sont *en minuscules* ;
- Les valeurs d'attributs doivent être *entre guillemets* ;
- Un document respectant ces critères est dit *bien formé* (well formed). Exemple:

```
<auteur nom="brillant" prenom="alexandre">...</auteur> <contact  
email='a@a.fr' />
```

IV. Support par les navigateurs

- **1. Famille Mozilla et Internet Explorer**

Ces navigateurs (à partir de la version 5 pour Internet Explorer) permettent l'affichage des documents XML sous la forme d'une arborescence dans le cas général ; si une feuille de style est spécifiée, le navigateur l'interprète (à partir de la version 6 pour Internet Explorer).

- **2. Google Chrome**

seules les versions égales ou supérieures à 6 affichent les fichiers XML à condition toutefois qu'une feuille de style XSL soit déclarée dans le prologue. Néanmoins son implémentation est encore partielle.

Exercice 1. Structuration d'informations 1

- XML permet de *structurer* une information. Il est donc nécessaire, avant d'envisager d'utiliser ce format, de se familiariser avec cette structuration.
- Le paragraphe suivant contient de l'information "en vrac". Réorganisez-la de manière à mettre en évidence sa structure logique, sans forcément passer par une mise en forme XML

Exercice 1. Structuration d'informations 1

- **Création d'un livre en XML**

On souhaite écrire un livre en utilisant le formalisme XML. Le livre est structuré en sections (au moins 2), en chapitres (au moins 2) et en paragraphes (au moins 2).

Le livre doit contenir la liste des auteurs (avec nom et prénom).

Tous les éléments doivent posséder un titre, sauf le paragraphe qui contient du texte.

Proposez une structuration XML de ce document (avec 2 auteurs, 2 sections, 2 chapitres par section et 2 paragraphes par chapitre).

Vérifiez, à l'aide de l'éditeur, que votre document est bien formé.

Attention : ne pas utiliser d'attributs ; l'encodage utilisé est ISO-8859-1

Votre document sera nommé **livre1.xml**.

Exercice 1. La codification

< paragraphes >

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

<paragraphe>Premier paragraphe</paragraphe>

<livre>

< paragraphe>Deuxième paragraphe</paragraphe
>

<titre>Mon livre</titre>

</paragraphes>

<auteurs> </chapitre>

<auteur><nom>Mohamed</nom><prenom>Alaoui< </chapters>

/prenom></auteur> </section>

<auteur><nom>Mustafa</nom><prenom>Arabi</p <section>

renom></auteur>

</auteurs>

<sections>

<section>

<titre>Section 1</titre>

<chapitres>

<chapitre>

<titre>Chapitre 1</titre>

<paragraphes>

<paragraphe>Premier paragraphe</paragraphe>

<paragraphe>Deuxième paragraphe</paragraphe>

</paragraphes>

</chapitre>

<chapitre>

<titre>Chapitre 2</titre>

<titre>Section 2</titre>

<chapitres>

< chapitre >

<titre>Chapitre 1</titre>

<paragraphes>

<paragraphe>Premier paragraphe</paragraphe>

<paragraphe>Deuxième
paragraphe</paragraphe>

</paragraphes>

</chapitre>

< chapitre >

<titre>Chapitre 2</titre>

<paragraphes>

<paragraphe>Premier paragraphe</paragraphe>

<paragraphe>Deuxième
paragraphe</paragraphe>

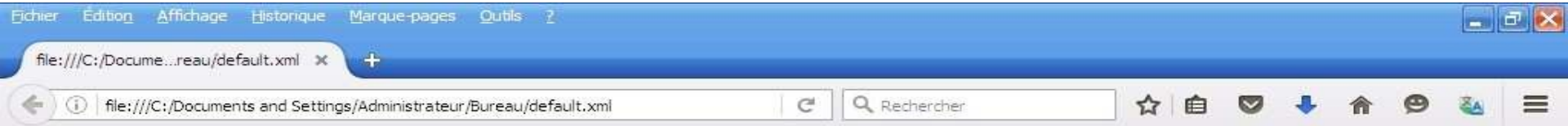
</paragraphes>

</chapitre>

</chapitres>

</section>

```
</sections>  
</livre>
```



Aucune information de style ne semble associée à ce fichier XML. L'arbre du document est affiché ci-dessous.

```
- <livre>
  <titre>Mon livre</titre>
  - <auteurs>
    - <auteur>
      <nom>Mohamed</nom>
      <prenom>Alaoui</prenom>
    </auteur>
    + <auteur></auteur>
  </auteurs>
  - <sections>
    - <section>
      <titre>Section 1</titre>
      - <chapitres>
        - <chapitre>
          <titre>Chapitre 1</titre>
          + <paragraphes></paragraphes>
        </chapitre>
        - <chapitre>
          <titre>Chapitre 2</titre>
          + <paragraphes></paragraphes>
        </chapitre>
      </chapitres>
    </section>
    + <section></section>
  </sections>
</livre>
```


Exercice 2

Utilisation des attributs Conception de livre2.xml à partir de livre1.xml

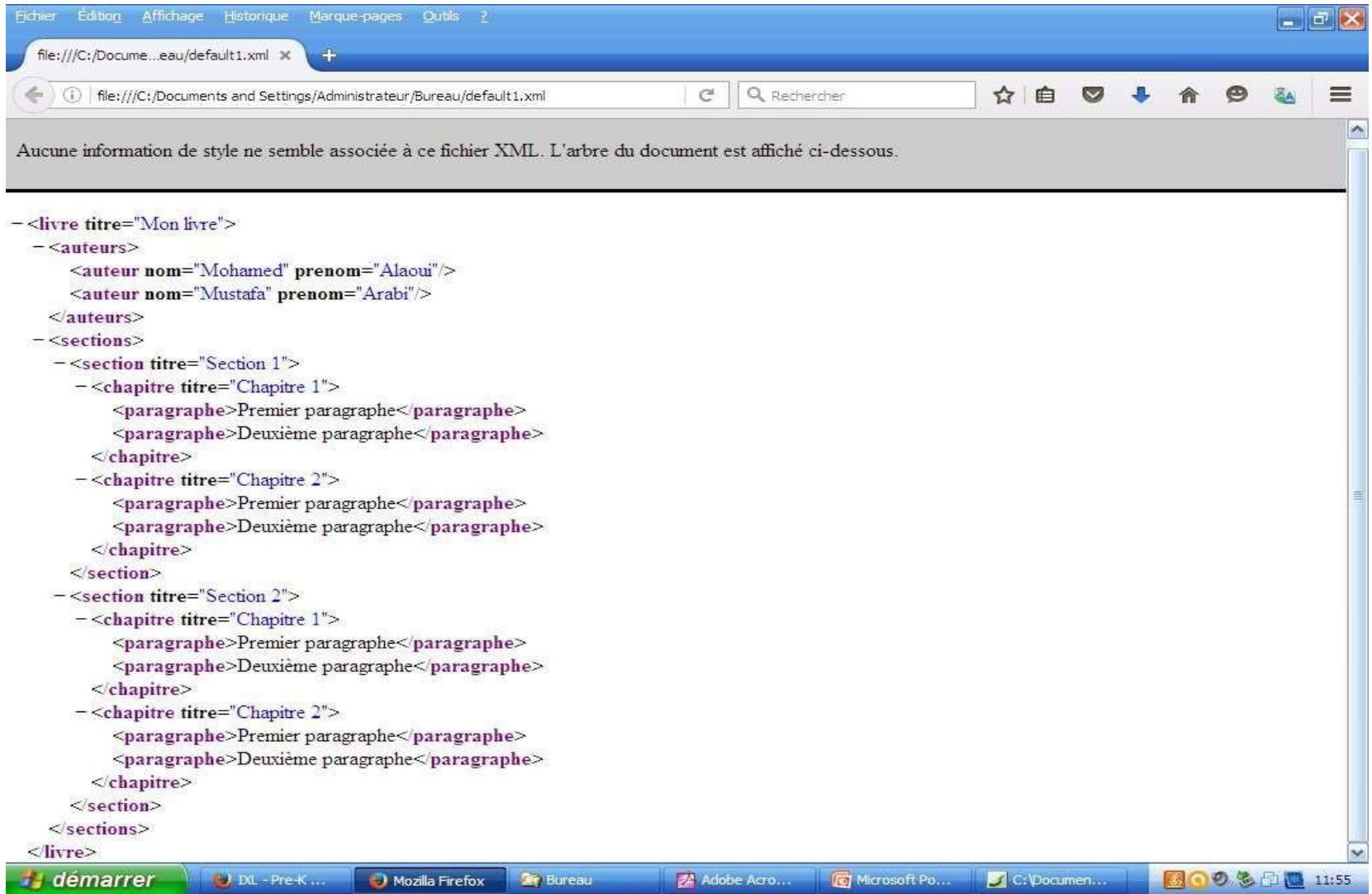
On souhaite compléter la structure du document XML de l'exercice précédent par les attributs nom et prenom pour les auteurs et titre pour le livre, les sections et les chapitres.

Analysez la structure du nouveau document. Y a-t-il des simplifications possibles ?

Vérifiez, à l'aide de l'éditeur, que votre document est bien formé.

Code exe:2

```
<?xml version="1.0" encoding="iso-8859-1"?>
<livre titre="Mon livre">
  <auteurs>
    <auteur nom="Mohamed" prenom="Alaoui"/>
    <auteur nom="Mustafa" prenom="Arabi"/>
  </auteurs>
  <sections>
    <section titre="Section 1">
      <chapitre titre="Chapitre 1">
        <paragraphe>Premier paragraphe</paragraphe>
        <paragraphe>Deuxième paragraphe</paragraphe>
      </chapitre>
      <chapitre titre="Chapitre 2">
        <paragraphe>Premier paragraphe</paragraphe>
        <paragraphe>Deuxième paragraphe</paragraphe>
      </chapitre>
    </section>
    <section titre="Section 2">
      <chapitre titre="Chapitre 1">
        <paragraphe>Premier paragraphe</paragraphe>
        <paragraphe>Deuxième paragraphe</paragraphe>
      </chapitre>
      <chapitre titre="Chapitre 2">
        <paragraphe>Premier paragraphe</paragraphe>
        <paragraphe>Deuxième paragraphe</paragraphe>
      </chapitre>
    </section>
  </sections>
</livre>
```



Affichage Résultat dans navigateur

DTD

Déclarations de **Types de** **Documents**

Plan : Partie DTD

- **Pourquoi écrire des définitions DTD?**

Vous vous demandez certainement à quoi servent ces définitions et pourquoi on les utilise, n'est-ce pas ?

Associer une définition à un document pour une bonne écriture de vos données XML. C'est important lorsque plusieurs personnes travaillent sur un même document.

Document Type Definition abrégé **DTD**

Les DTD : Les Types de DTD

- Une DTD peut être stockée dans deux endroits différents:
- Elle peut être incorporée au document XML (DTD *interne*), ou bien être un fichier à part (DTD *externe*). Cette dernière possibilité permet de la partager entre plusieurs documents XML.
- Il est possible de mêler DTD interne et externe.
- Il existe deux types de DTD externes : privé ou public. Les DTD privées sont accessibles uniquement en local (sur la machine de développement), tandis que les publiques sont disponibles pour tout le monde, étant accessibles grâce à

Les DTD : Les Types de DTD

un URI (Uniform Resource Identifier). • Une déclaration de type de document est de la forme :

<!DOCTYPE elt.racine SYSTEM "fichier.dtd">

- Cette déclaration se place juste après le prologue du document. L'élément racine du document XML rattaché à cette DTD est alors obligatoirement elt.racine.
- Le mot-clé SYSTEM est important et indique qu'il s'agit d'une DTD qui vous est propre.
- L'alternative est le mot-clé PUBLIC.

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Strict//EN" >

Les DTD : Les Types de DTD

- Exemple d'un fichier XML qui utilise la DTD « boncommande.dtd »

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

```
<!DOCTYPE commande SYSTEM "boncommande.dtd">
```

```
<commande>
```

```
<item>(...)</item>
```

```
<item>(...)</item>
```

```
<item>(...)</item>
```

```
</commande>
```

- La syntaxe DTD ne diffère pas entre une DTD interne et une externe.

Syntaxe

- les déclarations d'une DTD interne sont écrites à l'intérieur du prologue du document XML, celles d'une DTD externe sont stockées dans un fichier externe.

Les DTD : Les Types de DTD

- Exemple de déclarations pour une DTD interne au fichier XML:

```
<!DOCTYPE biblio [  
  <!ELEMENT biblio (livre)*>  
  <!ELEMENT livre (titre, auteur, nb_pages)>  
    <!ATTLIST livre type (roman | nouvelles | poemes | théâtre)  
      #IMPLIED lang CDATA "fr">  
  <!ELEMENT titre (#PCDATA)>  
  <!ELEMENT auteur (#PCDATA)>  
  <!ELEMENT nb_pages (#PCDATA)> ]>
```

- Exemple de déclarations pour une DTD externe au fichier XML:

```
<!DOCTYPE commande SYSTEM "boncommande.dtd">
```

DTD : Déclarations d'éléments

Déclaration élément simple

1.Généralités

- `<!ELEMENT balise (définition) >`
- le paramètre définition représente
- soit un type de données prédéfini,
- soit un type de données composé constitué lui-même d'éléments
- Types prédéfinis
- ANY: l'élément peut contenir tout type de donnée
- EMPTY: l'élément ne contient pas de données spécifiques (élément vide)
- #PCDATA: l'élément doit contenir une chaîne de caractères dans l'encodage courant (non interprétée par XML)
- Exemple
`<!ELEMENT Nom (#PCDATA)>`
`<Nom>Victor Hugo</Nom>`

Exemple

Exemple 1

```
<personne>  
  <nom>Mohamed Said</nom>  
</personne>
```

DTD Associe:

```
<!ELEMENT personne (nom)>  
<!ELEMENT    nom  
(#PCDATA)> Exemple 2:  
<!ELEMENT personne (nom)>  
<!ELEMENT nom EMPTY>
```

XML Associe:

```
<personne>  
  <nom ></nom>  
</personne>
```

NB: EMPTY et ANY, l'usage des parenthèses n'est pas obligatoire

Les DTD : Déclarations d'éléments

2. Élément texte

- Cet élément est le plus répandu, puisque c'est celui qui contient... du texte. Il se déclare ainsi :
- `<!ELEMENT elt (#PCDATA)>`

Les DTD : Déclarations d'éléments

3. Élément vide

- Un élément vide est, comme son nom l'indique, un élément qui n'a aucun contenu - que ce soit de type texte, ou bien un autre élément. Le mot-clé utilisé pour la déclaration de ce type d'élément est `EMPTY` :
`<!ELEMENT elt EMPTY>`
- Exemple : `<elt />`
- Un élément vide peut fort bien posséder un ou plusieurs attributs.
- Exemple:
``

La séquence des ELEMENTS

- <!ELEMENT balise (balise2, balise3, balise4, balise5, etc.)>

- Exemple: DTD

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

<!ELEMENT nom (#PCDATA)>

<!ELEMENT prenom (#PCDATA)>

<!ELEMENT age (#PCDATA)>

- XML associe:

<personne>

<nom>Mohamed</nom>

<prenom>Said</prenom>

<age>44</age>

</personne>

- **NB:** le DTD est invalide si les balises ne sont pas dans le bon ordre ou il manque une balise ou il y a une balise en trop, qui plus est non déclarée

La liste de choix

- `<!ELEMENT balise (balise2 | balise3 | balise4 | balise5 | etc.)>`
- Une **liste de choix** permet de dire qu'une balise contient l'une des balises décrites. Il suffit d'indiquer le nom des balises en les séparant par une **barre verticale**.

- DTD associe:

```
<!ELEMENT personne (nom | prenom)>
<!ELEMENT nom (#PCDATA)> <!ELEMENT
prenom (#PCDATA)>
```

- XML associe:

```
<personne>
  <nom>Mohamed</nom>
</personne>
```

OU

```
<personne>
  <prenom>Said</prenom> </personne>
```

NB:

les 2 balises prenom et nom ne peuvent pas être présentes en même temps. `<personne>`

```
  <prenom>Mohamed</prenom>
```

```
  <nom>Said</nom>
```

```
</personne>
```

il manque une balise

```
<personne ></personne>
```

La balise optionnelle

- Une balise peut être **optionnelle**. Pour indiquer qu'une balise est optionnelle, on fait suivre son nom par un **point d'interrogation**.
- `<!ELEMENT balise (balise2?, balise3, balise4)>`

• DTD Associe:

`<!ELEMENT personne (nom, prenom?)>`

`<!ELEMENT nom (#PCDATA)>`

`<!ELEMENT prenom (#PCDATA)>`

XML Associe:

`<personne>`

`<nom>Mohamed</nom>`

`</personne>`

Ou

`<personne>`

`<nom>Mohamed</nom>`

`<prenom>Said</prenom>`

`</personne>`

Faux si l'ordre des balises n'est pas respecté

`<personne>`

`<prenom>Said</prenom>`

`<nom>Mohamed</nom>`

`</personne>`

La balise répétée optionnelle

- Une balise peut être **répétée plusieurs fois** même si elle est optionnelle. Pour indiquer une telle balise, on fait suivre son nom par une **étoile**.

<!ELEMENT balise (balise2, balise3*, balise4)>

- DTD Associe: ____ <Agenda>

< personne >

<!ELEMENT Agenda (personne*)>

<nom>Mohamed</nom>

<!ELEMENT personne (nom, prenom)>

<prenom>Said</prenom>

<!ELEMENT nom (#PCDATA)>

</personne>

<!ELEMENT prenom (#PCDATA)>

</Agenda>

XML associe:

Ou

<Agenda>

<repertoire ></repertoire>

OU

<personne>

<Agenda>

<nom>Mohamed</nom> <personne>
<prenom>Said</prenom> <nom>Mohamed</nom>
</personne> <prenom>Said</prenom>
<personne> </personne>

<personne>

<nom>Hasna</nom>

<nom>Hasna</nom>

<prenom>Malak</prenom> </personne>

</personne> </Agenda>

</Agenda> Faux car il manque la balise prenom⁶⁰ dans

OU la seconde balise personne

La balise répétée

- Une balise peut être **répétée plusieurs fois**. Pour indiquer une telle balise, on fait suivre son nom par un **plus**.
- `<!ELEMENT balise (balise2, balise3+, balise4)>` DTD

Associe:

```
<!ELEMENT repertoire (personne+)>
<!ELEMENT personne (nom, prenom)>
<!ELEMENT nom (#PCDATA)> <!ELEMENT
prenom (#PCDATA)> XML Associe:
```

```
<repertoire>
  <personne>
    <nom>Mohamed</nom>
    <prenom>Said</prenom>
```

```
</personne>
<personne>
  <nom>Hasna</nom>
  <prenom>Malak</prenom>
</personne>
</repertoire>
```

OU

```
<repertoire>
  <personne>
    <nom>Mohamed</nom>
    <prenom>Said</prenom>
  </personne>
</repertoire>
```

Faux `< repertoire ></repertoire >`

Les DTD : Déclarations Éléments composés

Indicateurs d'occurrence

- Lors de la déclaration de séquence ou de choix d'éléments, à chaque élément enfant peut être attribuée une indication d'occurrence (?, + ou *).
- Exemples d'indicateur d'occurrences :

<!ELEMENT elto (elt1, elt2?, elt3+, elt4*)>

- elt1 ne comprend aucune indication d'occurrence. Il doit donc apparaître une seule et unique fois dans l'élément elto ;
- elt2 a pour indication d'occurrence ?. Cela signifie que l'élément doit apparaître au maximum une fois (il peut ne pas apparaître du tout) ;
- elt3 a pour indication d'occurrence +. Cela signifie que l'élément doit apparaître au moins une fois ;
- elt4 a pour indication d'occurrence *. Cela signifie que l'élément doit apparaître autant de fois que l'auteur le désire.

Éléments composés

- Élément composé d'une séquence ou d'un choix d'éléments
- Syntaxe spécifique avec opérateurs de composition d'éléments:

<!ELEMENT balise (composition) >

A et B	Explication	Exemples
A?	A (un seul) est une option, (donc: A ou rien)	<!ELEMENT personne (nom, email?)
A+	Il faut un ou plusieurs A	<!ELEMENT personne (nom, email+)
A*	A est une option, il faut 0, 1 ou plusieurs A	<!ELEMENT personne (nom, email*)
A B	Il faut A ou B, mais pas les deux	<!ELEMENT personne (email fax)
A , B	Il faut A suivi de B (dans l'ordre)	<!ELEMENT personne (nom, email ?)
(A, B)+	Les parenthèses regroupent. Ici, un ou plusieurs (A suivi de B)	<!ELEMENT liste (nom, email)+

Semi-structuré car on peut avoir un *mixed content*

- (#PCDATA | e1 | ... | en)

Exemple

- DTD

<!ELEMENT personne (nom, prenom+, tel?, adresse >

<!ELEMENT nom (#PCDATA) >

<!ELEMENT prenom (#PCDATA) >

<!ELEMENT tel (#PCDATA) >

<!ELEMENT email (#PCDATA) >

<!ELEMENT Adresse (ANY) >

- Document XML associé<personne>

<nom>Hugo</nom>

<prenom>Victor</prenom>

<prenom>Charles</prenom>

<tel>0383000000</tel>

<adresse><rue/><ville>Paris</ville></adresse>

</personne>

Les DTD : Déclarations d'éléments

Séquence d'éléments

Une séquence d'éléments est une liste ordonnée des éléments qui doivent apparaître en tant qu'éléments-enfants de l'élément que l'on est en train de définir.

Dans le fichier XML, ils doivent apparaître *dans l'ordre* de la séquence.

<!ELEMENT elto (elt1, elt2, elt3)>

- Exemple d'utilisation valide :

<elto>

<elt1>(...)</elt1>

<elt2>(...)</elt2>

<elt3>(...)</elt3>

</elto>

Les DTD : Déclarations d'éléments

Exemples d'utilisations non valides :

<elto>

<elt1>(...)</elt1>

<elt3>(...)</elt3>

</elto>

... car l'élément elt2 est manquant.

<elto>

<elt1>(...)</elt1>

<elt3>(...)</elt3>

<elt2>(...)</elt2>

</elto>

... car l'ordre des éléments n'est pas respecté.

Les DTD : Déclarations d'éléments

Choix d'éléments

Un choix d'élément donne... le choix dans une liste de plusieurs éléments possibles. L'utilisation précise dépend des indicateurs d'occurrence. De même que pour la séquence, les éléments-enfants doivent être déclarés dans la DTD. Cette liste est composée d'éléments séparés par le caractère | (combinaison de touches AltGr+6 sur un clavier AZERTY).

<!ELEMENT elto (elt1 | elt2 | elt3)>

Exemple d'utilisation valide :

**<elto><elt2>(...)</elt2></elto> Choix
d'éléments**

Les DTD : Déclarations d'éléments

Exemple d'utilisation non valide :

```
<elto>  
  <elt2>(...)</elt2>  
  <elt3>(...)</elt3>  
</elto>
```

Exemple d'utilisation d'un choix d'éléments avec
indicateurs d'occurrence par élément :

```
<!ELEMENT elto (elt1* | elt2* | elt3*)>
```

6. Choix d'éléments

Les DTD : Déclarations d'éléments

Exemple d'utilisation valide :

```
<elto>  
  <elt2>(...)</elt2>  
  <elt2>(...)</elt2>  
</elto>
```

Exemples d'utilisation non valide :

```
<elto>  
  <elt3>(...)</elt3>  
  <elt2>(...)</elt2>  
</elto>
```

```
<elto>  
  <elt2>(...)</elt2>
```

Les DTD : Déclarations d'éléments

<elt3>(…)</elt3> </elto>

6. Choix d'éléments

Exemple d'utilisation d'un choix d'éléments avec indicateur d'occurrence global :

<!ELEMENT elto (elt1 | elt2 | elt3)*> Exemple d'utilisation valide :

<elto>

<elt2>(…)</elt2>

<elt3>(…)</elt3>

<elt1>(…)</elt1>

</elto>

Les DTD : Déclarations d'éléments

Dans ce dernier cas, il n'y a pas de contrainte visible sur l'ordre d'apparition des éléments.

7. Élément quelconque

Il peut contenir tout autre élément défini dans la DTD, aussi bien qu'être vide ou contenir du texte.

Les éléments-enfants éventuels peuvent apparaître dans n'importe quel ordre, et en quantité non définie.

il est préférable de ne pas utiliser trop souvent ce type de déclaration, car on perd les avantages qu'offre la rédaction d'une DTD, qui sont de fixer des contraintes précises sur la structure du document XML qui lui est lié. Le mot-clef utilisé pour la déclaration de ce type d'élément est ANY.

Les DTD : Déclarations d'éléments

<!ELEMENT elt ANY>

8. Élément à contenu mixte

Un élément à contenu mixte peut contenir aussi bien du texte, que des éléments-enfants. Il se présente comme une liste de choix, avec des indicateurs d'occurrence bien choisis. Le texte contenu peut se trouver à n'importe quel endroit dans l'élément, et peut être une section CDATA.

Exemple de déclaration :

<!ELEMENT citation (#PCDATA | auteur)*>

Exemple d'utilisation :

<citation>

Être ou ne pas être <auteur>Shakespeare</auteur> </citation>

Quelques exemples

<!ELEMENT personne (nom_prenom | nom)>

<!ELEMENT nom_prenom (#PCDATA)>

<!ELEMENT nom (#PCDATA)>

- Cela nous autorise deux documents XML, soit :

<personne>

<nom_prenom>Brahim Ahmed</nom_prenom>

</personne>

- ou bien :

<personne>

<nom>Brahim</nom>

</personne>

Autre cas avec l'opérateur de séquence

<!ELEMENT personne(prenom,nom)>

<!ELEMENT prenom (#PCDATA)>

<!ELEMENT nom (#PCDATA)>

- Ici, l'opérateur de séquence limite les possibilités à un seul document XML valide :

<personne>

<prenom>Ahmed</prenom>

<nom>Brahim</nom>

</personne>

Les DTD :Déclarations d'attributs

1. Introduction

Comme on peut trouver dans un document XML des éléments possédant des attributs, il est normal que la DTD permette de définir des contraintes sur ces derniers. On peut déclarer et attacher à un élément donné chaque attribut séparément, mais il est préférable de les assembler sous la forme d'une liste. Chaque attribut défini dans la liste possède un nom et un type. On peut lui donner une valeur par défaut, ou bien le spécifier obligatoire. Le mot-clef de cette déclaration est ATTLIST.

Les DTD :Déclarations d'attributs

Pour indiquer que notre règle porte sur un **attribut**, on utilise le mot clef:
<!ATTLIST balise attribut type mode>

Une règle peut donc se diviser en 5 mots clefs :

ATTLIST, balise, attribut, type et mode.

l'attribut:

<personne sexe="masculin" /> DTD:
<!ATTLIST personne sexe type mode>

Type:

<!ATTLIST balise attribut (valeur 1 | valeur 2 | valeur 3 | etc.) mode>
<!ATTLIST personne sexe (masculin|féminin) mode> XML :
<personne sexe="masculin" ></personne> OU
<personne sexe="féminin" ></personne>

Mais pas

<personne sexe=« Animal" ></personne>

Les DTD :Déclarations d'attributs

- **attribut ayant pour type du texte non "parsé"**
- **non "parsé"** se cache en fait la possibilité de mettre ce que l'on veut comme valeur : un nombre, une lettre, une chaîne de caractères, etc. Il s'agit de données qui ne seront pas analysées par le "parseur" au moment de la validation et/ou l'exploitation de votre document XML.
- attribut contient du **texte non "parsé"**, on utilise la mot clef CDATA

<!ATTLIST balise attribut CDATA mode>

DTD:

<!ATTLIST personne sexe CDATA mode>

XML:

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

<personne sexe="féminin" ></personne>

<personne sexe="autre" ></personne>

<personne sexe="12" ></personne>

Les DTD: Déclarations d'attributs

2. Valeurs par défaut

Chaque attribut peut être requis, optionnel ou fixe et avoir une valeur par défaut. Les exemples suivants montrent la déclaration d'un attribut appelé attr attaché à un élément nommé elt.

Déclaration d'un attribut avec une valeur par défaut :

```
<!ELEMENT elt (...)>
```

```
  <!ATTLIST elt attr CDATA "valeur">
```

Un tel attribut n'est pas obligatoire. S'il est omis dans le fichier XML lors de l'utilisation de l'élément elt, il est considéré comme valant valeur.

Dans cet exemple, si on écrit <elt>(...)</elt>, cela est équivalent à écrire <elt attr="valeur">(...)</elt>.

Déclaration d'un attribut requis :

```
<!ELEMENT elt (...)>
```

```
  <!ATTLIST elt attr CDATA #REQUIRED>
```

Un tel attribut est obligatoire. Son absence déclenche une erreur du vérificateur syntaxique sur le fichier XML.

Les DTD : Déclarations d'attributs

- Déclaration d'un attribut optionnel :

<!ELEMENT elt (...)>

 <!ATTLIST elt attr CDATA #IMPLIED>

- Déclaration d'un attribut avec une valeur fixe :

<!ELEMENT elt (...)>

 <!ATTLIST elt attr CDATA #FIXED "valeur">

L'utilité d'un tel attribut peut sembler bizarre à première vue, puisqu'il ne peut prendre qu'une seule valeur. Cette fonctionnalité est cependant utile lors

Les DTD : Déclarations d'attributs

d'une mise à jour d'une DTD, pour préserver la compatibilité avec des versions ultérieures.

3. Type chaîne de caractères

Il s'agit là du type d'attribut le plus courant. Une chaîne de caractères peut être composée de caractères ainsi que d'entités analysables. Le mot-clef utilisé pour la déclaration de chaîne de caractère est CDATA.

Exemple de déclaration de CDATA :

```
<!ELEMENT elt (...)>
```

```
  <!ATTLIST elt attr CDATA #IMPLIED>
```

Exemples d'utilisations :

Les DTD : Déclarations d'attributs

```
<elt attr="Chaîne de caractères"></elt>  
<!ENTITY car "caractères">  
    <elt attr="Chaîne de &car;">(...)</elt>
```

4. Type ID

Ce type sert à indiquer que l'attribut en question peut servir d'*identifiant* dans le fichier XML. Deux éléments ne pourront pas posséder le même attribut possédant la même valeur.

Exemple de déclaration de type ID optionnel :

```
<!ELEMENT elt (...)>  
    <!ATTLIST elt attr ID #IMPLIED>  
<!ELEMENT elt1 (...)>  
    <!ATTLIST elt1 attr ID #IMPLIED>  
<!ELEMENT elt2 (...)>  
    <!ATTLIST elt2 attr ID #IMPLIED>
```

Les DTD : Déclarations d'attributs

La déclaration précédente interdit par exemple...

```
<elt attr="machin"></elt>  
    <elt2 attr="truc"></elt2>  
    <elt1 attr="machin"></elt1>
```

... ainsi que

```
<elt attr="machin"></elt>  
    <elt2 attr="machin"></elt2>  
    <elt1 attr="truc"></elt1>
```

5. Type énuméré

On peut parfois désirer limiter la liste de valeurs possibles pour un attribut. On le définit alors comme étant de type énuméré. Donner une autre valeur dans le fichier XML provoque une erreur.

Les DTD : Déclarations d'attributs

Exemple de déclaration d'une liste de choix d'attributs :

<!ELEMENT img EMPTY>

<!ATTLIST img format (BMP | GIF | JPEG) "JPEG">

Cet exemple déclare un attribut format d'un élément img. La valeur de cet attribut peut être BMP, GIF ou JPEG. Si aucune valeur n'est affectée à cet attribut, c'est la valeur par défaut qui le sera, ici JPEG. On notera l'absence de guillemets dans la liste des valeurs possibles. C'est une source courante d'erreur dans la rédaction d'une DTD.

Attributs résumé

- `<! ATTLIST balise attribut type mode >`
- *balise* spécifie l'élément auquel est attaché l'attribut
- *attribut* est le nom de l'attribut déclaré
- *type* définit le type de donnée de l'attribut choisi parmi:
 - CDATA: chaînes de caractères entre guillemets ("aa") non analysées
 - Enumération: liste de valeurs séparées par |
- `<! ATTLIST balise Attribut (Valeur1 | Valeur2 | ...) >`
- NMTOKEN: un seul mot
- ID et IDREF: clé et référence à clé
- *mode* précise le caractère obligatoire ou non de l'attribut
 - #REQUIRED: obligatoire (l'attribut doit être valué)
 - #IMPLIED: optionnel (l'attribut peut être valué)
 - #FIXED valeur: attribut avec valeur fixe (valeur fixée dans la DTD)

Exemple

```
<!ELEMENT personne (nom, prenom+, tel?, adresse >  
<!ELEMENT nom (#PCDATA) >  
<!ELEMENT prenom (#PCDATA) >  
<!ELEMENT tel (#PCDATA) >  
<!ELEMENT email (#PCDATA) >  
<!ELEMENT Adresse (ANY) >  
<! ATTLIST personne num ID, age CDATA, genre (Masculin | Feminin ) >  
<!ELEMENT auteur (#PCDATA) >  
<!ATTLIST auteur genre (Masculin | Feminin ) #REQUIRED  
ville CDATA #IMPLIED> <!ELEMENT editeur (#PCDATA) >  
<!ATTLIST editeur ville CDATA #FIXED "Paris">
```

ID et IDREF

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Document [
<!ELEMENT Document (Personne*)>
<!ELEMENT Personne (#PCDATA)>
<!ATTLIST Personne PNum ID #REQUIRED>
<!ATTLIST Personne Mere IDREF #IMPLIED>
<!ATTLIST Personne Pere IDREF #IMPLIED>
]>
```

Le XML correspondant au DTD

```
< Document >
< Personne PNum = "P1">Marie</PERSONNE>
< Personne PNum = "P2">Jean</PERSONNE>
< Personne PNum = "P3" Mere ="P1" Pere ="P2">Pierre</PERSONNE>
< Personne PNum = "P4" Mere ="P1" Pere ="P2">Julie</PERSONNE>
</Document >
```

Exercice 1

Utilisation d'une DTD

Créez la DTD carnet.dtd suivante :

```
<!ELEMENT carnet (personne+)>
```

```
<!ELEMENT personne EMPTY>
```

```
<!ATTLIST personne nom
```

```
CDATA #REQUIRED prenom
```

```
CDATA #IMPLIED telephone
```

```
CDATA #REQUIRED>
```

- Créez un document XML qui soit valide par rapport à cette DTD.

Correction

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

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

```
<carnet>
```

```
<personne nom="dupont" prenom="jean"  
telephone="001122"/>
```

```
<personne nom="dupond" telephone="221100"/>
```

```
</carnet>
```

Afficher le XML avec CSS

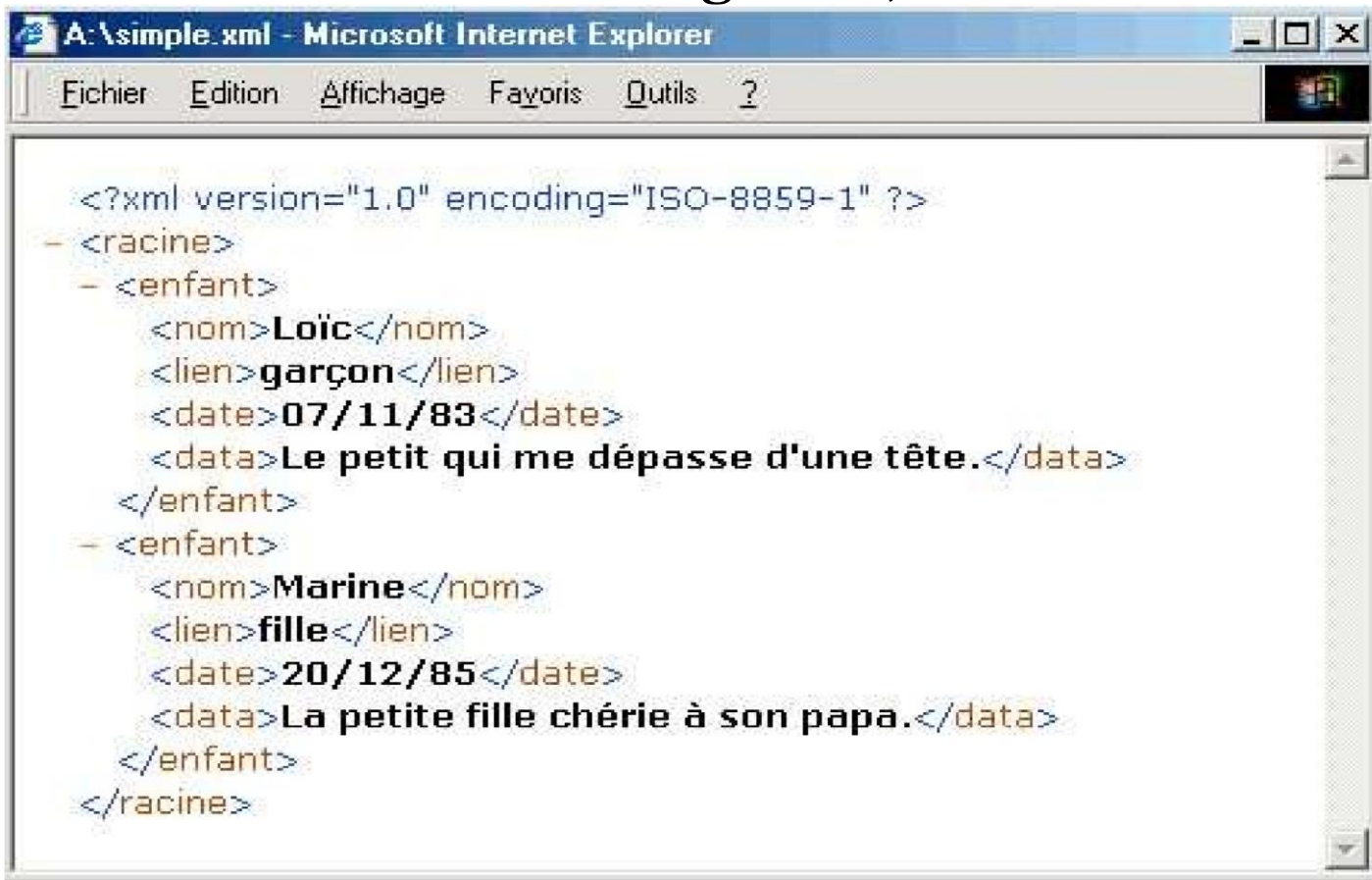
- **Le CSS**
- Pour afficher les balises XML, on peut faire appel aux feuilles de style (CSS), comme pour le style dans Html.
- A chaque balise "inventée" dans le fichier XML, on va définir un élément de style que le navigateur pourra alors afficher.
- Voir l'exemple suivant:

Un exemple de XML + CSS

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

<racine>
<enfant>
<nom>Loïc</nom>
<lien>garçon</lien>
<date>07/11/83</date>
<data>Le petit qui me dépasse d'une tête.</data>
</enfant>
<enfant>
<nom>Marine</nom>
<lien>fille</lien>
<date>20/12/85</date>
<data>La petite fille chérie à son papa.</data>
</enfant>
</racine>

Affiché dans le navigateur, cela nous donne

A screenshot of a Microsoft Internet Explorer window. The title bar reads "A:\simple.xml - Microsoft Internet Explorer". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", "Outils", and "?". The main content area displays an XML document with the following structure:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <racine>
  - <enfant>
    <nom>Loïc</nom>
    <lien>garçon</lien>
    <date>07/11/83</date>
    <data>Le petit qui me dépasse d'une tête.</data>
  </enfant>
  - <enfant>
    <nom>Marine</nom>
    <lien>fille</lien>
    <date>20/12/85</date>
    <data>La petite fille chérie à son papa.</data>
  </enfant>
</racine>
```

On ajoute un fichier .css dont voici le contenu

```
<style type="text/css">
```

```
racine , enfant {} nom
```

```
{
```

```
display: block; width: 250px; font-  
size: 16pt ; font-family: arial ; font-  
weight: bold; background-color: teal;  
color: white; padding-left: 10px;
```

```
} lien { display: block; font-size:  
12pt; padding-left:  
10px;
```

```
}
```

```
date { display:  
block; font-size:  
12pt; color: red ;  
font-weight:  
bold; padding-  
left: 10px;
```

```
}
```

```
data {  
displa
```

```
y:
```

```
block;
```

```
font-
```

```
size:
```

```
11pt ;
```

```
font-
```

```
style:
```

```
italic;
```

```
font-
```

```
family
```

```
: arial
```

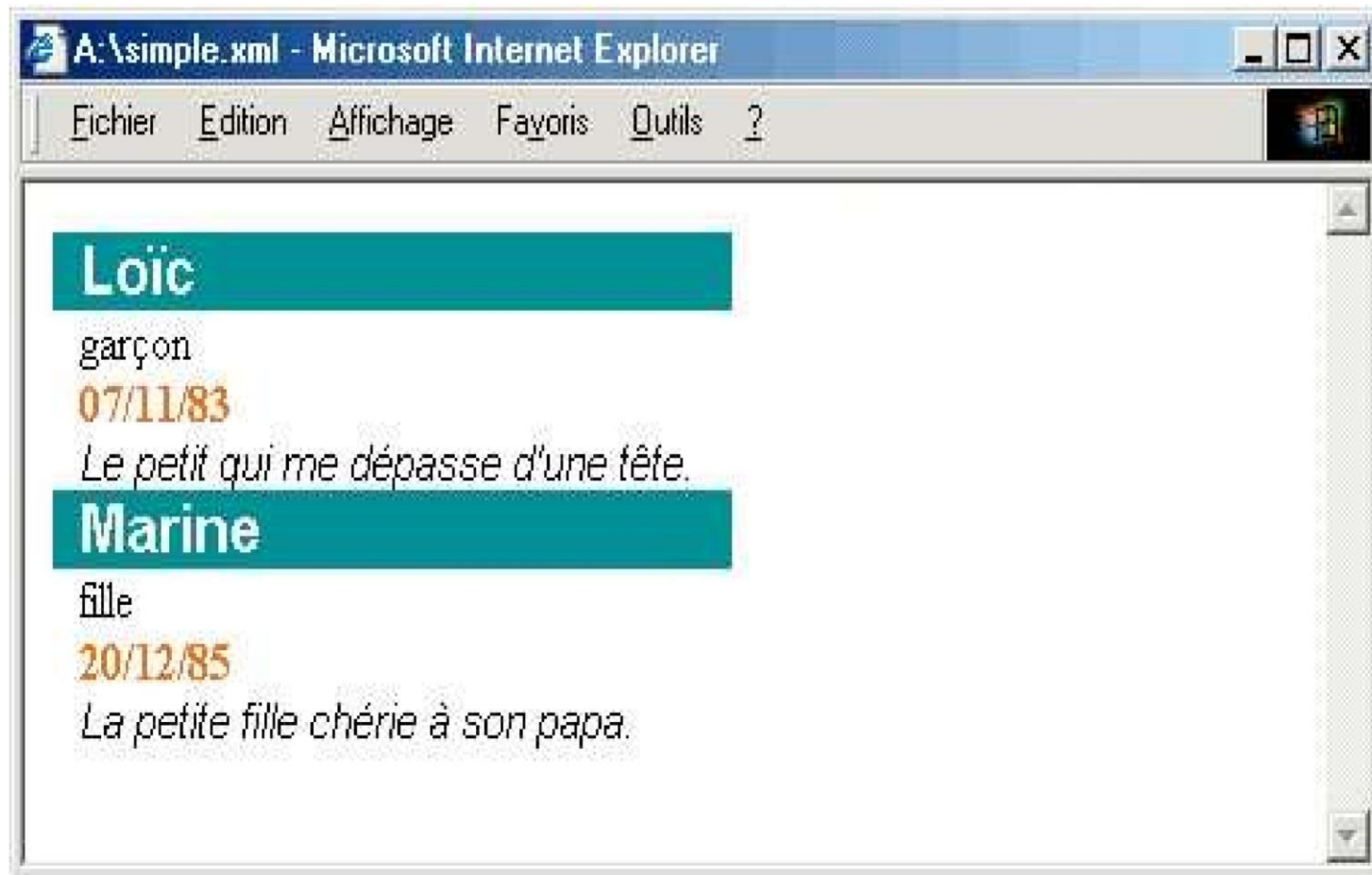
```
;
```

```
padding-left: 10px;
```

```
}
```

```
</style>
```

Le Résultat



Biensur Après avoir ajouté un lien vers le fichier css dans le fichier xml :
<?xmlstylesheet href="css.css" type="text/css"?>

Afficher du XML dans Html

Du XML dans un fichier Html

On peut toujours incorporer du XML dans un fichier Html avec la balise `<xml> ... </xml>`.

Mais logique, quand les navigateurs rencontrent des balises incorrectes ou inconnues, rien n'est affiché.

Ce sera le cas avec vos balises XML incorporées dans un fichier Html.

Heureusement, on peut passer par une astuce qui répond au doux nom romantique de "îlots de données" [data islands].

- **Les Data Islands [les îles de données]**

Dans un fichier Html, vous pouvez créer un " îlot" de données se trouvant dans un fichier XML distinct et en extraire des données que vous pouvez alors afficher dans le document Html.

Dans le fichier Html, désigner le fichier xml extérieur avec un identifiant id :

```
<xml id="fichierxml" src="simple.xml"></xml>  
[datasrc="#id"]
```

L'essentiel XML HTML

- . Dans un tableau Html, que l'on relie par un attribut à la source des données au moyen de l'identifiant désigné
- . [datasrc="#id"],
- . on peut reprendre des données du fichier XML avec l'attribut de champ de donnée qui reprend comme valeur le nom de la balise XML [datafld="balise_xml"].

Le code HTML

```
<html> <body>
```

Voici du Html...

```
<xml id="fichierxml" src="simple.xml"></xml>
```

```
<table border="1" datasrc="#fichierxml"> <tr>
```

```
<td><span datafld="nom"></span></td>
```

```
<td><span datafld="lien"></span></td>
```

```
<td>Anniversaire le <span datafld="date"></td> </tr>
```

```
</table>
```

Et voici encore du Html !

```
</body>
```

```
</html>
```

Afficher du XML dans Html

Voici du Html...

Loïc	garçon	Anniversaire le 07/11/83
Marine	fil le	Anniversaire le 20/12/85

Et voici encore du Html !

Les feuilles de styles XSL

Plan de la partie: Feuilles de Styles

- 1. Introduction**
- 2. Structure d'un document XSL**
- 3. Exemple simple**
- 4. Exercice 5: Mise en forme simple à l'aide de XSL**
- 5. Exemple avec boucle**

Le langage XSL

- Le XSL pour e**X**tensible **S**tylesheet **L**anguage ou "langage extensible de feuilles de style" est une recommandation du W3C datant de novembre 1999.
- C'est donc un standard dans le domaine de la publication sur le Web.
- Le XSL est en quelque sorte le langage de feuille de style du XML.
- Un fichier de feuilles de style reprend des données XML et produit la présentation ou l'affichage de ce contenu XML selon les souhaits du créateur de la page.

Les feuilles de style

1. Introduction

- XSL signifie *eXtensive Stylesheet Langage*, ou langage extensible de feuille de style. XSLT signifie *eXtensible Stylesheet Langage Transformation*.
- Comme son nom l'indique, le XSL ne sert pas uniquement de langage de feuille de style.
- Il est aussi un très puissant manipulateur d'éléments. Il permet de transformer un document XML source en un autre.
-

Les feuilles de style

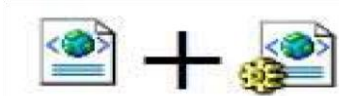
2. Structure d'un document XSL

- La structure de base d'un document XSL commence par un *prologue*, puis un élément `<xsl:stylesheet>` pouvant contenir quelques attributs, notamment une déclaration d'espace de noms ainsi que le numéro de version. L'exemple suivant présente l'appel à cet élément tel que nous le pratiquerons dans ce cours :
- **`<?xml version="1.0" encoding="ISO-8859-1"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> (...)</xsl:stylesheet>`**
- L'élément `<xsl:stylesheet>` est l'élément racine du document XSL. C'est lui qui contient tous les modèles, y compris celui qui est associé à la racine du document XML, modèle que l'on note `<xsl:template match="/">`. L'attribut `match="/"` indique que ce modèle s'applique à la racine du document XML.

XML + XSL

- Le XSL est donc le complément indispensable pour l'affichage du XML. D'où notre titre :

XML + XSL.



- Voici le fichier XSL :

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<xsl:value-of select="demoXML/message"/>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Le résultat dans le navigateur est alors :

Voici du XML.

Que peut faire encore XSL

- **Le XSL ne fait pas qu'afficher !**

Le XSL ne permet pas uniquement l'affichage de XML. Il permet aussi :

- de sélectionner une partie des éléments XML.
- de trier des éléments XML.
- de filtrer des éléments XML en fonction de certains critères.
- de choisir des éléments.
- de retenir des éléments par des tests conditionnels.

Les feuilles de style

1. Exemple simple

- Il est possible de traiter de manière simple un fichier XML contenant une information relativement linéaire.

Ainsi, l'exemple déjà présenté d'une composition de bouteille d'eau, dans le cas d'une seule bouteille, se prête facilement à une simple mise en forme HTML.

Premier Exemple Les feuilles de style XSL

- Exemple d'un document XML lié à une feuille de style XSL simple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="bouteille1.xsl"?>
<bouteille>
  <marque>Sidi Harazem</marque>
  <composition>calcium 71mg/l, magnésium 5,5mg/l, chlorure
20mg/l, nitrate 1mg/l, traces de fer.</composition>
  <source>
    <ville>Sidi Harazem la Source</ville>
    <departement>Fes</departement>
  </source>
  <code_barre>3274080005003</code_barre>
  <contenance>150cl</contenance>
  <ph>7,45</ph>
</bouteille>
```

La Feuille XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <head><title>Exemple de sortie HTML</title>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
</head>
<body>
  <h1>Bouteille de marque <xsl:value-of select="bouteille/marque"/></h1> <h2>Composition:</h2>
  <p><xsl:value-of select="bouteille/composition"/></p>
  <h2>Lieu d'origine:</h2>
  <p>Ville de <b>
<xsl:value-of select="bouteille/source/ville"/></b>, dans le département
<b><xsl:value-of select="bouteille/source/departement"/></b></p>
  <h2>Autres informations</h2> <ul> <li>Contenance: <xsl:value-of
select="bouteille/contenance"/></li>
  <li>pH: <xsl:value-of select="bouteille/ph"/></li> </ul> </body> </html> </xsl:template>
</xsl:stylesheet>
```

Le langage XSL

- le langage XML et XSL vous passez à une vitesse supérieure.

- Le XML et son complément le XSL
- **Le XML ne fait rien. Il faudra passer par le XSL !**
- le Html a été conçu pour afficher de l'information, le XML a été créé pour structurer de l'information. Il ne fait rien d'autre !

<?xml version="1.0"?>

<demoXML>

<message>Voici du XML</message> </demoXML>

```
<?xml version="1.0" ?>
- <demoXML>
  <message>Voici du XML</message>
</demoXML>
```

Les Langages de XSL

- Le XSL comporte en fait 3 langages :
- Le XSLT qui est un langage qui Transforme un document XML en un format, généralement en Html, reconnu par un navigateur.
- Le Xpath qui permet de définir et d'adresser des parties de document XML.
- Le XML Formatter pour "formater" du XML (transformé) de façon qu'il puisse être rendu sur des PCpockets ou unités de reconnaissance vocale.
- dans ce cours, nous nous limiterons au XSLT et Xpath.

XSL est dérivé du XML

- Le langage XSL est un langage de balises dérivé du langage XML. Le XSL reprend donc toutes les règles de syntaxe du XML (détaillée dans la partie relative au XML).
- **les balises sensibles à la casse, s'écrivent en minuscules.**
- **toutes les balises ouvertes doivent être impérativement fermées.**
- **les balises vides auront aussi un signe de fermeture soit <balise/>.**
- **les balises doivent être correctement imbriquées.**
- **les valeurs des attributs doivent toujours être mises entre des guillemets.**
- **le document XSL devra être "bien formé" [Well-formed].**

Formatage du Document XSL 1

- **Principe de fonctionnement**

- Pour la petite histoire 1999 fait référence à l'année d'apparition du concept XSL.

1. Le document XSL reprend donc la structure et la syntaxe de n'importe quel document XML.

2. le document XSL comporte un document Html qui sera quant à lui reconnu par le navigateur et qui servira de support à tout ou partie des données du document XML associé.

3. XSL fonctionne avec une ou plusieurs "**templates**", sorte de gabarit pour définir comment afficher des éléments du fichier XML. Les éléments concernés du fichier XML sont déterminés par l'attribut "**match**".

- **<?xml version="1.0"?>**

Le XSL est dérivé du XML. Il est normal que le document XSL commence par la déclaration de document XML, soit <?xml version="1.0"?>.

- **<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">**

La seconde ligne déclare que le document est du XSL *extensible stylesheet*.

L'attribut xmlns fait référence au "namespace" utilisé. Le namespace officiel du W3C est xmlns:xsl="http://www.w3.org/1999/XSL/Transform".

document XSL 2

- **<xsl:template match="/">**

La balise template et son attribut match.

Cette balise template va déterminer un gabarit dans lequel on va transformer des éléments du fichier XML sous une forme que le navigateur pourra afficher.

Les éléments du fichier XML sont déterminés par l'attribut match="/". Le slash / entre guillemets signale que sont concernées toutes les balises XML du document associé à partir de la racine [root].

- **<html>**

- **<body>**

Début de la partie Html qui servira de support pour l'affichage du document dans le navigateur. Attention, balises en minuscules !

- **Diverses balises Html et XSL... Par exemple :**

- **<xsl:value-of select="chemin d'accès/élément"/>**

La balise <xsl:value-of> sera fréquemment utilisée car elle permet de sélectionner un élément du fichier XML associé pour le traiter dans le fichier XSL.

Dans l'attribut select, on détermine le chemin d'accès vers la balise XML souhaitée (puisque le XML est structuré) comme le chemin d'accès de répertoire en sous-répertoire vers un dossier.

Attention, on utilise bien ici le "forward slash" soit / .

document XSL 3

- **</body>**
- **</html>**

Fin de la partie en Html.

- **</xsl:template>**

La fermeture de la balise de template.

- **</xsl:stylesheet>**

Le document XSL se termine obligatoirement par la fermeture de la balise de déclaration de document XSL.

Attention ! Pour que ce fichier XSL soit d'une quelconque utilité, il faut encore faire référence, **dans le fichier XML** au fichier XSL.

- On ajoutera donc dans le fichier XML :
- **<?xml-stylesheet type="text/xsl" href="nom_du_fichier_xsl.xml"?>** Cette balise indique au navigateur qu'une feuille de style [stylesheet] est associé au fichier XML et qu'il doit aller chercher le fichier de style à l'adresse indiquée par l'attribut href.

Commentaires

- Les commentaires existent aussi en XSL. Voici la balise en question :
- `<xsl:comment>`
- Commentaire
- `</xsl:comment>`

Exercice . Mise en forme simple à l'aide de XSL

- Créer le fichier XML suivant :

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

```
<recette>
```

```
  <entete>
```

```
    <auteur>Casimir</auteur>
```

```
    <titre>Recette du Gloubi-Boulga</titre>
```

```
    <remarque>Pour une personne</remarque> </entete>
```

```
<procedure>
```

Remplir un saladier avec de la confiture de fraises, du chocolat râpé, des bananes écrasées, de la moutarde forte, des saucisses de Toulouse écrasées tièdes mais crues. Mélanger

vigoureusement jusqu'à obtenir une bouillie marron-clair. Il est normal qu'il y ait des grumeaux. Les proportions sont environ égales pour tous les ingrédients, mais il est possible de varier selon les goûts de chacun. </procedure>

```
</recette>
```

Exercice . Mise en forme simple à l'aide de XSL

- Créer une feuille de style XSL permettant à partir de cette fiche recette de produire une page HTML qui :
 - a pour titre le contenu de la balise titre ;
 - commence par un titre `<h1>` ayant comme contenu le contenu de l'élément titre ;
 - donne ensuite le nom de l'auteur de la recette ;
 - affiche ensuite le mot **Remarque** : puis le contenu de l'élément remarque ;
 - affiche **Procédure** en niveau `<h2>` ;
 - dans un paragraphe, présente la procédure à suivre.

Solution : Exercice 5. Mise en forme simple à l'aide de XSL

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:template match="/">
  <html>
    <head>
      <title>
        <xsl:value-of select="//titre" />
      </title>
    </head>
    <body>
      <h1> <xsl:value-of select="//titre" /> </h1>
      <p> <b>Auteur:</b> <xsl:value-of select="//auteur" /> </p>
      <p> <b>Remarque:</b> <xsl:value-of select="//remarque" /> </p>
      <h2>Procédure</h2>
      <p><xsl:value-of select="//procedure" /> </p>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Le but de l'exercice est de représenter la compilation de mp3 sous forme d'un tableau.

```
<?xml version="1.0"?>
<compilation>
  <mp3>
    <titre>Foule sentimentale</titre>
    <artiste>Alain Souchon</artiste>
  </mp3>
  <mp3>
    <titre>Solaar pleure</titre>
    <artiste>MC Solaar</artiste>
  </mp3>
  <mp3>
    <titre>Le baiser</titre>
    <artiste>Alain Souchon</artiste>
  </mp3>
  <mp3>
    <titre>Pourtant</titre>
    <artiste>Vanessa Paradis</artiste>
  </mp3>
  <mp3>
    <titre>Chambre avec vue</titre>
    <artiste>Henri Salvador</artiste>
  </mp3> </compilation>
```

Passons maintenant au fichier XSL

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

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

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

```
<html>
```

```
<body>
```

```
<table border="1" cellspacing="0" cellpadding="3">
```

```
<tr bgcolor="#FFFF00">
```

```
<td>Titre</td>
```

```
<td>Artiste</td>
```

```
</tr>
```

```
<tr>
```

```
<td><xsl:value-of select="compilation/mp3/titre"/></td>
```

```
<td><xsl:value-of select="compilation/mp3/artiste"/>
```

```
</td> </tr> </table>
```

Après les balises de départ d'un fichier XSL, on aborde un `</body>` tableau tout à fait classique en Html. On remplit la cellule `< /html >` correspondante au titre par la balise **xsl:value-of** avec l'attribut

`</xsl:template>` **select="compilation/mp3/titre"** qui indique comme `</xsl:stylesheet>` chemin d'accès la balise racine compilation la balise mp3 la balise titre. Et bien entendu de même pour la balise artiste.

XML plein de balises devient un beau tableau sympathique.



Afficher toutes les données

On va reprendre le fichier XSL auquel on va ajouter la balise **xsl:for-each** [pour chaque] avec comme attribut **select="compilation/mp3"**.

Pour chaque arborescence où l'on retrouve les balises compilation et mp3, il suffit de faire une ligne de tableau **<tr>** avec dans la premier cellule **<td>**, le contenu de la balise **<titre>** **xsl:value-of select="titre"** et dans la seconde cellule **<td>**, le contenu de la balise **<artiste>** **xsl:value-of select="artiste"**. Ce qui donne:

```
<?xml version='1.0'?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="1" cellspacing="0" cellpadding="3">
<tr bgcolor="#FFFF00">
<td>Titre</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<tr>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:for-each>
</table> </body>
</html>
</xsl:template>
</xsl:stylesheet>
```



Titre	Artiste
Foule sentimentale	Alain Souchon
Solaar pleure	MC Solaar
Le baiser	Alain Souchon
Pourtant	Vanessa Paradis
Chambre avec vue	Henri Salvador

-

Autres Possibilités de XSL

Le langage XSL est plus qu'une série de balises pour afficher du XML.

- Il comporte aussi des possibilités plus qu'utiles quand on est confronté à des données. Nous verrons comment le XSL permet :
- **de trier les données XML en ordre croissant ou décroissant.**

- - **de filtrer des éléments XML en fonction de certains critères.**
 - **de choisir des éléments.**
 - **de retenir des éléments par des tests conditionnels.**

Trier avec le langage XSL

Le langage XSL permet en quelques mots de trier des données du fichier XML associé en ordre croissant ou décroissant. Ainsi, il suffit

- d'ajouter l'attribut **order-by="+balise"** pour trier en ordre croissant et **order-by="-balise"** pour trier en ordre décroissant. Et c'est tout !
- souligner avec cette balise, la puissance du langage XSL. le langage XSL est assez intuitif. Ainsi order-by="+balise" peut se lire :
- "ordonner ou trier par ordre croissant (+) les données comprises entre la balise désignée".

-

Trier avec le langage XSL

Nous allons trier notre compilation de mp3 en XML en ordre alphabétique croissant du nom des artistes.

- pour changer un peu, on permute les colonnes "titre" et "artiste" pour bien montrer que le XSL affiche les données du fichier XML selon le fichier Html (ou autre) qu'il contient.

Trier avec le langage XSL

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="1" cellspacing="0" cellpadding="3">
<tr bgcolor="#FFFF00">
<td>Artiste</td>
<td>Titre</td>
</tr>
<xsl:for-each select="compilation/mp3" order-by="+artiste">
<tr>
<td><xsl:value-of select="artiste"/></td>
<td><xsl:value-of select="titre"/></td>
</tr>
</xsl:for-each>
</table>
</body> </html>
</xsl:template>
</xsl:stylesheet>
```



Artiste	Titre
Alain Souchon	Foule sentimentale
Alain Souchon	Le baiser
Henri Salvador	Chambre avec vue
MC Solaar	Solaar pleure
Vanessa Paradis	Pourtant

Choisir avec le XSL Conditionnel

La balise `<xsl:if> ... </xsl:if>` permet d'effectuer un choix dans les données du fichier XML.

On ajoutera l'attribut `match` où l'on indique l'élément choisi. Ce qui en résumé donne : **`<xsl:if match=".[balise='xxx']"> balises Html </xsl:if>`**

XSL IF

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="1" cellspacing="0" cellpadding="3">
<tr bgcolor="#FFFF00">
<td>Titre</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<b><xsl:if match=".[artiste='Vanessa Paradis']">
<tr>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</b>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```



C:\WINDOWS\Bureau\xml_if.xml - Microsoft Int...



Fichier Edition Affichage Favoris Outils ?



Titre	Artiste
Pourtant	Vanessa Paradis

Exemple 2 conditionnelles

- Voici la balise en question :
- `<xsl:if test="">`
- `</xsl:if>`
- Par exemple, le code suivant affichera « J'ai Super Mario » mais pas « J'ai Metroid Prime », sauf si vous l'ajoutez dans le fichier XML.
- `<xsl:for-each select="test/jeu">`
- `<xsl:if test="nom = 'Mario Super ">`
- `<p>J'ai SuperMario !</p>`
- `</xsl:if>`
- `<xsl:if test="nom = 'Super Man`
- `<p>J'ai Super man!</p">>`
- `</xsl:if>`
- `</xsl:for-each>`

Filtrer avec le langage XSL

Le langage XSL permet aussi de filtrer les données du fichier XML associé selon des critères comme égal, pas égal, plus grand que, plus petit que. Pour ce faire, il suffira d'utiliser l'attribut `select="chemin_d'accès[balise='xxx']"`.

Les opérateurs possibles sont :

- `=` pour égal.
- `!=` pour différent (non égal).
- `>` pour plus grand que.
- `<` pour plus petit que.

Dans la compilation mp3, ne reprenons que les titres de l'artiste Alain Souchon. L'attribut `select` devient **`select="compilation/mp3[artiste='Alain Souchon']"`**.

Filtrer avec le langage XSL

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="1" cellspacing="0" cellpadding="3">
<tr bgcolor="#FFFF00">
<td>Titre</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3[artiste='Alain Souchon']">
<tr>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:for-each>
</table>
</body> </html>
</xsl:template>
</xsl:stylesheet>

```



Choix conditionnels et le XSL

- Le XSL permet de faire un choix conditionnel par la balise
- **<xsl:choose>**. A l'intérieur de cette balise, on peut déterminer une action lorsque une condition est vérifiée **xsl:when** et dans le cas contraire prévoir une autre action **xsl:otherwise**.
- Exemple d'utilisation :

```
<xsl:choose>  
<xsl:when test=".[annee!='"']"> (On affiche année uniquement quand il n'est pas vide) <tr>  
<td width="100px">Année </td>  
<td> <xsl:value-of select="./annee" /> </td>  
</tr>  
</xsl:when>  
</xsl:choose>
```

Les feuilles de style

Exemple avec boucle

Il arrive que la structure du document XML ne soit pas linéaire, mais fondée sur, par exemple, des boucles, ou bien comporte un nombre indéterminé de fois un même élément ; c'est d'ailleurs le plus souvent le cas.

On peut ainsi reprendre l'exemple de la bouteille d'eau, qui se présente sous la forme du fichier XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
  <?xml-stylesheet type="text/xsl" href="bouteille1.xsl"?>  
  <bouteille>  
    <marque>Cristaline</marque>  
    <composition>
```


Les feuilles de style

```
<ion_positif>calcium 71mg/l</ion_positif>
<ion_negatif>nitrate 1mg/l</ion_negatif>
<ion_positif>magnésium 5,5mg/l</ion_positif>
<ion_negatif>chlorure 20mg/l</ion_negatif>
<autres_materiaux>fer</autres_materiaux>
</composition>
<source>
  <ville>St-Cyr la Source</ville>
  <departement>Loiret</departement>
</source>
<code_barre>3274080005003</code_barre>
<contenance>150cl</contenance>
```

Les feuilles de style

<ph>7,45</ph>

</bouteille>

- **Cette fois-ci, il faut tenir compte d'un nombre a priori indéterminé d'éléments ion_positif,**
- **par exemple. Il suffit pour cela d'introduire dans la feuille de style un élément**
- **<xsl:for-each select="ce_qu_on_cherche_a_afficher"/>, qui permet de faire une boucle sur l'élément cherché :**

XML avec Javascript

- Exemple

Réaliser des transformations XSLT avec PHP

- Pour réaliser des transformations XSLT avec PHP, il faut que le module `php_xsl` soit installé. Si vous avez installé PHP avec l'installateur Windows, vous ne disposerez pas de ce module par défaut : il vous faut utiliser le fichier ZIP (vous pouvez par exemple écraser le répertoire `php` par le contenu de ce fichier).

Réaliser des transformations XSLT avec PHP

```
<html>
<body>
<?
$doc = new DOMDocument();
$doc->load( "carnet.xml" );
$xsl = new XSLTProcessor();
$xslt = new DOMDocument();
$xslt->load( "carnet.xslt" );
$xsl->importStyleSheet( $xslt ); echo
$xsl->transformToXML($doc);
?>
</body>
</html>
```

Comme vous pouvez le constater, nous utilisons deux arborescences DOM : l'une pour le document XML et l'autre pour la feuille de styles. Nous nous sommes contentés de mettre en sortie le résultat de la transformation XSLT.

Les schémas

Plan de la partie : Schémas XML

- Limitations des DTD
- Avantages des schémas
- Les schémas : Structure de base
- Les schémas: Déclarations d'éléments
- Les schémas: Déclarations d'attributs
- Les schémas: Les types de données
- Exercice 8: Déclaration d'éléments
- Exercice 9: Déclarations d'attributs

Les schémas

1. Limitations des DTD

- Premièrement, les DTD ne sont pas au format XML. Cela signifie qu'il est nécessaire d'utiliser un outil spécial pour "parser" un tel fichier, différent de celui utilisé pour l'édition du fichier XML.
- Deuxièmement, les DTD ne supportent pas les "espaces de nom" (nous reviendrons sur cette notion)

En pratique, cela implique qu'il n'est pas possible, dans un fichier XML défini par une DTD, d'importer des définitions de balises définies par ailleurs.

- Troisièmement, le "typage" des données est extrêmement limité

Avantages des schémas

XML Schema propose des nouveautés en plus des fonctionnalités fournies par les DTD :

- Le typage des données est introduit, ce qui permet la gestion de booléens, d'entiers, d'intervalles de temps... Il est même possible de créer de nouveaux types à partir de types existants.
- La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément. C'est sans aucun doute l'innovation la plus intéressante de XML Schema.
- Le support des espaces de nom.
- Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif (rappel : dans une DTD, on était limité à 0, 1 ou un nombre infini d'occurrences pour un élément).
- Les schémas sont très facilement concevables par modules.

Les schémas : 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  
  chema"> <!-- déclarations d'éléments,  
  d'attributs et de types ici -->  
</xsd:schema>
```

- L'élément racine est l'élément **xsd:schema**.

Les schémas:Déclarations d'éléments

Un élément, dans un schéma, se déclare avec la balise `<xsd:element>`. Par exemple,

```
<?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> <xsd:element  
      name="remarque" type="xsd:string"></xsd:element>  
    <!-- déclarations de types ici -->  
  </xsd:schema>
```

Le schéma précédent déclare deux éléments : * •

- un élément contacts et un élément remarque.
- Chaque élément est "typé" -c'est-à-dire qu'il doit respecter un certain format de données. L'élément contacts est ainsi du type typeContacts, qui est un type complexe défini par l'utilisateur. L'élément remarque quant à lui est du type xsd:string qui est un type simple prédéfini de XML Schema.

Les schémas: Déclarations d'attributs

a. Déclaration simple

À la différence des 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. De plus, les déclarations d'attributs doivent être placées *après* les définitions des types complexes, autrement dit, après les éléments `<xsd:sequence>`, `<xsd:choice>` et `<xsd:all>` (voir plus loin). Pour mémoire, rappelons que dans une DTD, l'ordre des déclarations n'a pas d'importance.

L'exemple suivant montre la déclaration d'un attribut mais de type `xsd:date` (un autre type simple) qui indique la date de dernière mise à jour de la liste des contacts.

```
<?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 -->
```

Les schémas: Déclarations d'attributs

```
<xsd:complexType name="typeContacts">
  <!-- déclarations du modèle de contenu ici -->
  <xsd:attribute name="maj" type="xsd:date" />
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

b. Contraintes d'occurrences

Tout comme dans une DTD, un attribut peut avoir un indicateur d'occurrences.

L'élément attribut 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...). Par exemple, la ligne suivante permet de rendre l'attribut maj optionnel, avec une valeur par défaut au 11 octobre 2003 s'il n'apparaît pas (le format de date est standardisé : cette date s'écrit donc à l'anglo-saxonne 2003/10/11 ; cela permet en outre de plus facilement classer les dates).

Les schémas: Déclarations d'attributs

```
<xsd:attribute name="maj" type="xsd:date" use="optional"  
  default="200310-11" />
```

Quand l'attribut fixed est renseigné, la seule valeur que peut prendre l'attribut déclaré est celle de l'attribut fixed. Cet attribut permet de "réserver" des noms d'attributs pour une utilisation future, dans le cadre d'une mise à jour du schéma.

c. Regroupements d'attributs

XML Schéma propose une fonctionnalité supplémentaire, permettant de déclarer des groupes d'attributs (groupes auxquels il est possible de faire appel lors d'une déclaration d'éléments). Cela permet d'éviter de répéter des informations de déclarations.

d. Déclaration d'élément ne contenant que du texte avec un (ou plusieurs) attribut(s)

Un tel élément est de type complexe, car il contient au moins un attribut. Afin de spécifier qu'il peut contenir également du texte, on utilise l'attribut mixed de l'élément `<xsd:complexType>`. Par défaut, `mixed="false"`; il faut dans ce cas forcer `mixed="true"`. Par exemple,

Les schémas: Déclarations d'attributs

```
<xsd:element name="elt">  
  <xsd:complexType mixed="true">  
    <xsd:attribute name="attr" type="xsd:string" use="optional" />  
  </xsd:complexType> </xsd:element>
```

3. Déclaration et référencement

- Il est recommandé de commencer par déclarer les éléments et attributs de type simple, puis ceux de type complexe. On peut en effet faire "référence", dans une déclaration de type complexe, à un élément de type simple préalablement défini. Par exemple...

```
<xsd:element name="pages" type="xsd:positiveInteger" />  
  <xsd:element name="auteur" type="xsd:string" />  
  <xsd:element name="livre">  
    <xsd:complexType>  
      <xsd:sequence>  
        <xsd:element ref="auteur" />
```

Les schémas: Déclarations d'attributs

```
<xsd:element ref="pages" />  
</xsd:sequence>  
</xsd:complexType>  
</xsd:element>
```

Les schémas: Les types de données

1. Introduction

XML Schema permet de spécifier des types de données bien plus finement que le langage DTD. Il distingue notamment types simples et types complexes.

2. Types simples

a. Généralité

Les types de données simples ne peuvent comporter ni attributs, ni éléments enfants. Il en existe de nombreux, prédéfinis, mais il est également possible d'en "dériver" de nouveaux Enfin, il est possible de déclarer des "listes" de types.

Les types de données les plus simples (les chaînes de caractères) que permettaient les DTDs sont conservés, mais d'autres ont fait leur apparition. On pourra envisager, par exemple, dans un schéma décrivant un bon de commande, la déclaration d'un attribut quantité :

```
<xsd:attribute name="quantite" type="xsd:positiveInteger" use="default" value="1" />
```


Les schémas: Les types de données

... qui force la valeur de l'attribut à un être un entier positif. Un bon de commande XML suivant ce schéma, ayant une commande spécifiant `quantite="-3"` sera alors automatiquement refusé par le système. Un tel document peut dès lors être enregistré directement dans une base de données, sans contrôle supplémentaire sur ce point. **c.**

Listes

Les types listes sont des suites de types simples (ou *atomiques*). XML

Schéma possède trois types de listes intégrés : NMTOKENS, ENTITIES et IDREFS. Il est également possible de créer une liste personnalisée, par "dérivation" de types existants. Par exemple,

```
<xsd:simpleType name="numéroDeTéléphone">  
  <xsd:list itemType="xsd:unsignedByte" />  
</xsd:simpleType>
```

Un élément conforme à cette déclaration serait `<téléphone>01 44 27 60 11</téléphone>`.

3. Les types complexes

Les schémas: Les types de données

a. Introduction

Un élément de type simple ne peut contenir pas de sous-élément. Il est nécessaire pour cela de le déclarer de type "complexe". On peut alors déclarer, des séquences d'éléments, des types de choix ou des contraintes d'occurrences

b. Séquences d'éléments

Nous savons déjà comment, dans une DTD, nous pouvons déclarer un élément comme pouvant contenir une suite de sous-éléments dans un ordre déterminé. Il est bien sûr possible de faire de même avec un schéma.

On utilise pour ce faire l'élément `xsd:sequence`, qui reproduit l'opérateur , du langage DTD. Ainsi...

Les schémas: Les types de données

```
<xsd:complexType>  
  <xsd:sequence>  
    <xsd:element name="nom" type="xsd:string" />  
    <xsd:element name="prénom" type="xsd:string" />  
    <xsd:element name="dateDeNaissance" type="xsd:date" />  
    <xsd:element name="adresse" type="xsd:string" />  
    <xsd:element name="adresseElectronique" type="xsd:string" />  
    <xsd:element name="téléphone" type="numéroDeTéléphone" />  
  </xsd:sequence>  
</xsd:complexType>
```

... est équivalent à une déclaration d'élément, dans une DTD, où apparaîtrait (nom, prénom, dateDeNaissance, adresse, adresseElectronique, téléphone).

c. Choix d'élément

Les schémas: Les types de données

On peut vouloir modifier la déclaration de type précédente en stipulant qu'on doive indiquer soit l'adresse d'une personne, soit son adresse électronique. Pour cela, il suffit d'utiliser un élément `xsd:choice` :

```
<xsd:complexType name="typePersonne"> <sequence>
  <xsd:element name="nom" type="xsd:string" />
  <xsd:element name="prénom" type="xsd:string" />
  <xsd:element name="dateDeNaissance" type="xsd:date" />
  <xsd:choice>
    <xsd:element name="adresse" type="xsd:string" />
    <xsd:element name="adresseElectronique" type="xsd:string" />
  </xsd:choice>
</sequence>
  <xsd:element name="téléphone" type="numéroDeTéléphone" />
</xsd:complexType>
```

Ce connecteur a donc les mêmes effets que l'opérateur `|` dans une DTD. **d.**

L'élément all

Les schémas: Les types de données

Cet élément est une nouveauté par rapport aux DTD. Il indique que les éléments enfants doivent apparaître une fois (ou pas du tout), et dans n'importe quel ordre. Cet élément `xsd:all` doit être un enfant direct de l'élément `xsd:complexType`. Par exemple...

```
<xsd:complexType>
  <xsd:all>
    <xsd:element name="nom" type="xsd:string" />
    <xsd:element name="prénom" type="xsd:string" />
    <xsd:element name="dateDeNaissance" type="xsd:date" />
    <xsd:element name="adresse" type="xsd:string" />
    <xsd:element name="adresseElectronique" type="xsd:string" />
    <xsd:element name="téléphone" type="numéroDeTéléphone" />
  </xsd:all>
</xsd:complexType>
```

... indique que chacun de ces éléments peut apparaître une fois ou pas du tout (équivalent de l'opérateur ? dans une DTD), et que l'ordre des éléments n'a pas d'importance (cela n'a pas d'équivalent dans une DTD).

Les schémas: Les types de données

Les fils de l'élément `xsd:all` doivent impérativement apparaître au plus une fois, ce qui signifie que deux attributs, que nous allons voir maintenant, doivent être renseignés. Ces attributs sont `minOccurs` et `maxOccurs`.

f. Création de type complexe à partir de types simples

Il est possible également de créer un type complexe à partir d'un type simple. On peut avoir besoin de définir un élément contenant une valeur simple, et possédant un attribut, comme `<poids unite="kg">67</poids>`, par exemple. Un tel élément ne peut pas être déclaré de type simple, car il contient un attribut. Il faut *dériver* un type complexe à partir du type simple `positiveInteger` :

```
<xsd:complexType name="typePoids">
  <xsd:simpleContent>
    <xsd:extension base="xsd:positiveInteger">
      <xsd:attribute name="unite" type="xsd:string" />
    </xsd:extension>
  </xsd:simpleContent>
```

Les schémas: Les types de données

</xsd:complexType>

L'élément <xsd:simpleContent> indique que le nouvel élément ne contient pas de sous-élément.

Exercice 8. Déclaration d'éléments

Rédiger un Schema XML pour une bibliographie. Cette bibliographie :

- contient des livres et des articles ;
- les informations nécessaires pour un livre (élément livre) sont :
 - son titre général (élément titre) ;
 - les noms des auteurs (éléments auteur) ;
 - ses tomes (élément tomes) et pour chaque tome (éléments tome), leur nombre de pages (élément pages) ;
 - des informations générales sur son édition (élément infosEdition) comme par exemple le nom de l'éditeur (élément editeur), le lieu d'édition (élément lieuEdition), le lieu d'impression (élément lieuImpression), son numéro ISBN (élément ISBN) ;
- les informations nécessaires pour un article (élément article) sont :
 - son titre (élément titre) ;
 - les noms des auteurs (éléments auteur) ;
 - ses références de publication (élément infosPublication) : nom du journal (élément nomJournal), numéro des pages (élément pages), année de publication (élément anneePublication) et numéro du journal (élément numéroJournal)
- on réservera aussi un champ optionnel, pour chaque livre et chaque article, pour un avis (élément avis) personnel.
- Tester ce Schema XML avec un fichier XML .

Exercice 9. Déclarations d'attributs

Modifier le Schéma précédent... On ne déclarera, pour le moment, que des types de chaînes de caractères.

- ... en ajoutant un attribut optionnel soustitre à l'élément titre ;
- ... en faisant de l'élément tome un élément vide et en lui ajoutant un attribut requis nbPages et un attribut optionnel sousTitre ;
- ... en faisant de l'élément nomJournal un attribut de l'élément infosPublication et en lui donnant comme valeur par défaut Feuille de Chou ;
- Utiliser ce Schéma pour créer un fichier XML valide.

Documents bien formés /documents valides

XML comporte la notion de document bien formé et de document valides

- Un document bien formé : répond aux règles de base de XML
- Balises (éléments) ouvrantes /fermantes avec texte au milieu
- Balises « vides »
- Pas d'imbrication (un élément ne peut pas contenir un élément du même nom)
- Un document valide
- Doit se conformer à une DTD ou un schéma
- Doit respecter exactement la DTD ou le schéma (ne pas changer l'ordre des éléments, par exemple)

Comparaison DTD/Schémas

Il arrive souvent que l'on ait besoin de définir un type de document (figer l'ensemble des éléments et attributs possibles)

- DTD
 - Document type definition – Hérité de SGML
 - Syntaxe différent de XML
 - Pas vraiment de typage des éléments (on ne peut pas dire que l'élément année contient un entier)
 - Compris par tous les parseurs/vérificateurs
- Schéma
 - écrit en XML (même syntaxe qu'un document XML)
 - Permet un typage fin des éléments
 - Pas encore beaucoup de vérificateurs disponibles