Technologie XML

A. Zinedine

Plan

A. Zinedine

Introduction

- XML, C'est quoi ?
- Domaine d'application
- Outils logiciels

A. Zinedine

3

C'est Quoi?

- XML : eXtensible Markup Language
  - C'est un langage de Marquage
  - IL est extensible
  - Très strict
  - Met l'accent sur la structure des documents

A. Zinedine

4

Langage de Marquage

- Langage de Marquage = langage se basant sur la notion de balisage
- Marquage ou Balisage = du code inséré dans un document texte. IL stocke les informations nécessaire pour le traitement du document

Markup Language

- Exemples
  - Marquage procédural
  - Marquage générique
  - SGML : Standard General Markup Lang

SGML

- C'est un standard de l'ISO
- C'est un métalangage : il permet de définir d'autres langages de balisage
  - HTML est une application de SGML

HTML : le succès du Web

- C'est un langage destiné spécialement pour le web
- Très simple,
- Très tolèrent
- Derrière le succès du web

HTML : limité !

- N'est plus un petit langage
  - Un nombre énorme de balise
  - Autres langages complémentaires
- IL est malgré tout très limité
  - IL ne répond pas à tous les besoins
  - IL mélange le contenue, la structure et la présentation

Extensibilité

- Les applications Web ont de plus en plus de nouveaux besoins et bien spécifiques:
  - Commerce : balises pour les produits, les prix, les clients…
  - Moteurs de recherches : balises pour les mots clés, pour la description,…
  - Mathématiques : balises pour les formules, les théorèmes, …
  - …

HTML

- Ajouter de nouvelles balises ne résout pas le problème: on aura toujours besoin d'autres
- De plus : certaines applications préfèrent un petit langage plutôt qu'un grand langage (par exemple: l'accès au web par les téléphones mobiles)

Solution ?

- Comment donc obtenir un langage avec peu de balise et qui répond à tous les besoins?

  L'extensibilité

Extensibilité

- C'est la capacité de créer son propre jeu de balises
- XML: pas de balises prédéfinies
- XML est donc un métalangage : Vous pouvez créer vos propres langages de balisages

XML est Très strict

En effet :

- La tolérance de HTML à un prix:
  - Des navigateurs énormes et coûteux
- La tolérance de HTML n'est plus très importante:
  - Les utilisateurs utilisent de plus en plus les éditeurs graphiques

XML et les document structuré

- **Exemple…**

XML et les documents structurés

- **XML est un « ensemble de standards » pour échanger et publier l'information d'une manière structurée.**
- **XML est utilisé pour décrire et manipuler les documents structurés**

XML et les documents structurés

- XML structure les document sous forme d'arborescence , mais il n'impose pas comment peupler cette arborescence : il est donc très flexible
- XML fournit un mécanisme pour manipuler à la fois l'information et sa structure sous-jacente.

XML et les documents structurés

- XML offre des mécanisme pour manipuler automatiquement l'information par des application.
- Cette manipulation passe bien sûr à travers la structure

Application de XML

- Application de documents
  - Manipulation des informations destinées pour les humains
- Application de données
  - Manipulation des informations destinées pour les logiciels

Domaines d'application

- Publication
- Commerce électronique
- Gestion électronique du contenu
- Échange de données entre Systèmes d'informations hétérogènes
- ...

Les Outils Logiciels

- Navigateurs XML
- Éditeurs XML
- Parseurs XML
- Processeurs XSL

La syntaxe XML

- XML impose un certain nombre de règles de syntaxe
- XML est très strict en ce qui concerne le respect de ces règles et ne tolère la moindre erreur
- Un document qui respect toutes les règle de syntaxe est dit un document bien formé (well formed)
- Un document qui n'est pas bien formé ne peut être traité comme étant du XML

```
<?xml version = "1.0"  ?>
<!– exemple de document XML -->
<liste>
        <étudiant code ="E1">
                <nom>Moujtahid</nom>
                <prénom>Moujidd</prénom>
                <age>25</age>
                <adresse>25, rue Najah, Fès</adresse>
                <photo source ="moujtahid.jpg" />
        </étudiant>
        <étudiant code ="E2">
                <nom>Kaddouri</nom>
                <prénom>Kaddour</prénom>
                <age>26</age>
                <adresse>26, rue Al Falah, Fès</adresse>
                <photo source ="kaddouri.jpg" />
        </étudiant>
        <étudiant code ='E3'>
                <nom>Jallouli</nom>
                <prénom>Jalloul</prénom>
                <age>27</age>
                <photo source ="jallouli.jpg" />
        </étudiant>
</liste>
```

23

La syntaxe XML

- Éléments
  - Les noms en XML
  - La casse
  - Les attributs
  - Les éléments vides
- Les chevauchements
- L'élément racine
- Les commentaires
- La déclaration XML
- Les entités

A. Zinedine

24

Les éléments XML

- <balise> Contenu </balise>

Les noms XML

- Les noms des éléments doivent commencer par : a, b, c,… ,A, B, C,… ou _.
- Le reste du nom est constitué par des lettres, des chiffres, _ , - , . , : .
- Les nom en XML ne doivent pas commencer par xml
- Les espaces ne sont pas autorisés
- Le «:» a une signification particulière. Éviter de l'utiliser

La casse

- La casse est importante dans les noms XML
  - <ETUDIANT>
  - <etudiant>
  - <EtuDiant>
- Par convention, en préfère le minuscule pour les noms des éléments et des attributs

Attributs

- Pour ajouter une information supplémentaire aux éléments

<élément attribut = "valeur " >
Contenu
</élément>

- Les mêmes règles de nommage que les éléments

Fermeture des balises et éléments vides

- Toute balise ouverte doit être fermée
- Même les balises des éléments vides (qui ne contiennent pas de contenu) doivent être fermés:

Exemple:

```
<photo source="moujtahid.jpg"></photo>
```

Ou plus simple et plus préféré:

```
<photo source="moujtahid.jpg" />
```


Les chevauchements

- Les éléments en XML doivent être proprement imbriqués: il faux éviter les chevauchement:

YES [(……)] |||| [(……]) NO

YES <b1><b2> …</b2></b1>||||<b1><b2>… </b1></b2> NO

Elément racine

- Tout document XML doit être une arborescence proprement construite
- Donc, tout document XML possède un seul élément de niveau supérieur : la racine de l'arbre

Les commentaires

- Les commentaires en XML ont la même syntaxe qu'en HTML:

<!– C'est un commentaire -->

La déclaration XML

- La déclaration XML (ou prologue) est :

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

- La déclaration XML est optionnelle est doit être sur la première ligne du document

- On peut y déclarer la version de XML, l'encodage utilisé (par défaut utf-8), …

```
<?xml version ="1.0" encoding = "utf-16" ?>
```


Les entités

- Certains caractères entrent dans le codage du XML et sont considérés comme spéciaux
- XML propose des « entités prédéfinies» pour permettre d'introduire ces caractères proprement dans le contenu d'un document XML:

«<» = «<»
«>» = «>»
«"» = «"»
«'» = «'»
«&» = «&»

- L'utilisateur peut créer ses propres entités

Les espaces de noms

- Parmi les objectifs de XML: faciliter la collaboration et l'échange de données entre collaborateurs
- Dans cet esprit: on se trouve souvent avec des documents XML combinant plusieurs langages définis par plusieurs auteurs
- On peut se trouver avec des éléments différents avec le même nom
- L'ambiguïté et les erreurs peuvent donc se générer

Les espaces de noms

- Exemple

```
<?xml version="1.0" ?>
<biblio>
   <livre>
      <titre>Technologie XML</titre>  <!– le titre d'un livre, élément du langage biblioML -->
      <auteur>
         <titre>Dr.</titre>   <!– balise issue d'un autre langage XML qui permet de définir des personne-->
         <nom>Hakim Fahim</nom>
      </auteur>
   </livre>

            …
            …
</biblio>
```

Les espaces de noms

- Il faut donc trouver un moyen pour assurer l'unicité de chaque nouveau élément défini dans le monde entier
- Solution: préfixer les déférents éléments

```
<?xml version="1.0" ?>
<biblio>
   <livre>
      <L1:titre>Technologie XML</L1:titre>  <!– nous choisissons ici le préfixe L1 (pour language1) -->
      <auteur>
         <L2:titre>Dr.</L2:titre>   <!-- … et le préfixe L2 (pour language2) -->
         <nom>Hakim Fahim</nom>
      </auteur>
   </livre>

            …
            …
</biblio>
```


Les espaces de noms

- Solution: préfixer les déférents éléments
- Mais, Quel préfixe choisir? Il se peut que deux auteurs différents choisissent le même nom d'éléments et même préfixes !
- Solution: Chaque organisme utilisera son nom de domaine (déjà unique!)
- Exemple: je peux créer proprement l'élément suivant:

  <http://www.fsdm.usmba.ac.ma/zinedine:liste>

Les espaces de noms

- Inconvénients: les noms d'éléments sont ainsi trop longs
- Solution: déclarer les espaces de nom et utiliser des préfixes raccourcis:

```
<?xml version="1.0" ?>
<za:liste  xmlns:za="http://www.fsdm.usmba.ac.ma/zinedine" >
    <za:étudiant code ="E1">
            <za:nom>Moujtahid</za:nom>
            <za:prénom>Moujidd</za:prénom>
            <za:age>25</za:age>
            <za:adresse>25, rue Najah, Fès</za:adresse>
            <za:photo source ="moujtahid.jpg" />
    </za:étudiant>
    …
    …
</za:liste>
```


Les espaces de noms

- Dans la réalité toutes les éléments sont préfixés par:

http://www.fsdm.usmba.ac.ma/zinedine

- L'utilisateur se contente de les préfixer par « za »
- Le parseur se charge de la traduction en interne

```
<?xml version="1.0" ?>
<za:liste  xmlns:za="http://www.fsdm.usmba.ac.ma/zinedine" >
    <za:étudiant code ="E1">
            <za:nom>Moujtahid</za:nom>
            <za:prénom>Moujidd</za:prénom>
            <za:age>25</za:age>
            <za:adresse>25, rue Najah, Fès</za:adresse>
            <za:photo source ="moujtahid.jpg" />
    </za:étudiant>
    …
    …
</za:liste>
```

Les espaces de noms

- Avant de pouvoir utiliser un préfixe: vous devez le déclarer
- Pour déclarer un espace de nom, dans la balise ouvrante d'un élément mettez:

  xmlns:préfixe = "nom de domaine choisi"
- Tous les enfants de cet élément reconnaissent le préfixe

Les schémas

- Les schémas
  - DTD (Document Type Definition)
  - XSD (XML Schema Definition)

Les DTDs

- DTD = Document Type Defintion
- C'est un outil pour définir le schéma de notre langage :
  - Quelles sont les balises autorisé?
  - Dans quel ordre?
  - Nombre d'occurrences autorisées de chaque balises
  - Quels attribut?
  - ….

Les DTD

- Les DTD sont héritées du SGML
- Elles permettent de définir un langage d'une manière formelle
- Compréhensible par l'homme et par la machine
- Une DTD n'est pas obligatoire pour un document XML, mais, il est recommandé de définir le schéma de votre document pour pouvoir vérifier sa conformité
- Le parseur est le composant logiciel chargé de valider un document XML = vérifier sa conformité à une DTD
- Un document XML qui respecte les règles définie dans une DTD est dit valide

```
<?xml version="1.0" ?>
<!DOCTYPE liste [
<!ELEMENT photo EMPTY >
<!ELEMENT CV EMPTY >
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT tél (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT étudiant (nom,prénom,age,tél*,email,photo,(description|CV)?)>
<!ELEMENT liste (étudiant+)>

<!ATTLIST liste année CDATA #FIXED "2018" >
<!ATTLIST étudiant code ID #REQUIRED>
<!ATTLIST tél type (fixe|portable) "portable" >
<!ATTLIST photo source CDATA #REQUIRED
                                    largeur CDATA #IMPLIED
                                    hauteur CDATA #IMPLIED>
<!ATTLIST CV file CDATA #REQUIRED >
]>
<liste>
        <étudiant code ="E1">
                <nom>Moujtahid</nom>
                <prénom>Moujidd</prénom>
                <age>25</age>
                <adresse>25, rue Najah, Fès</adresse>
                <photo source ="moujtahid.jpg" />
        </étudiant>
```

45

Les DTD

- Une DTD est donc un ensemble de déclarations textuelle qui définissent la structure de votre langage XML
- Les DTD permettent à plusieurs collaborateurs de se mettre d'accord facilement sur le format des données échangées

A. Zinedine

46

Les DTD

- Une DTD peut être soit interne soit externe:
  - Interne : incorporée dans le document XML
  - Externe : écrite dans un fichier à part
    - Soit privée (dans un fichier du système de l'utilisateur)
    - Soit publique (dans une URL publique accessible)

Les DTDs

- Une DTD est déclarée avec la déclaration

  <!DOCTYPE racine>

  - Une DTD interne est incorporée entre crochets:

    <!DOCTYPE racine […….]>
  - Une DTD externe privée est déclarée comme suit:

    <!DOCTYPE racine SYSTEM "fichier.dtd">
  - Une DTD externe publique est déclarée comme suit:

    <!DOCTYPE racine PUBLIC "FPI" "url vers dtd">

Les DTD

- Dans la déclaration on précise la racine du langage
- C'est logique car le parseur doit connaitre le point d'entrée à votre document XML

Exemple de déclaration de DTD interne

```
<?xml version="1.0" ?>
<!DOCTYPE liste [
<!ELEMENT photo EMPTY >
<!ELEMENT nom (#PCDATA)>
...
....
]>
<liste>
        ...
        ...
        ...
<liste>
```

Exemple de déclaration de DTD Externe privée

```
<?xml version="1.0" ?>
<!DOCTYPE liste SYSTEM  "listeEtudiants.dtd"  >
<liste>
    <étudiant code ="E1">
          <nom>Moujtahid</nom>
          <prénom>Moujidd</prénom>
          <age>25</age>
          <adresse>25, rue Najah, Fès</adresse>
          <photo source ="moujtahid.jpg" />
    </étudiant>
    …
    …
</liste>
```

Document XML

```
<!ELEMENT photo EMPTY >
<!ELEMENT CV EMPTY >
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT tél (#PCDATA)>
…
…
<!ATTLIST liste année CDATA #FIXED "2018" >
<!ATTLIST étudiant code ID #REQUIRED>
<!ATTLIST tél type (fixe|portable) "portable" >
<!ATTLIST CV file CDATA #REQUIRED >
….
```

listeEtudiant.dtd

déclaration d'une DTD Externe publique

- **Une DTD externe publique est déclarée comme suit:**
  <!DOCTYPE racine PUBLIC "FPI" "url vers la dtd">
  - **racine: la racine du document XML**
  - **PUBLIC : pour indiquer que la DTD est publique**
  - **FPI: Formal Public Identifier, manière standard pour nommer les DTD publiques**
    - **±//propriétaire//DTD Label//XX**
    - **± : (+) pour langage standardisé par l'ISO, (-) si non**
    - **propriétaire : le nom du propriétaire du langage**
    - **label: une brève description du langage**
    - **XX: la langue du langage**
  - **Exemple:**

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3c.org/TR/1999/loose.dtd">

Les DTDs: Déclaration des éléments

- Syntaxe: <!ELEMENT nom-élément contenu-élément >
- Contenu-élément:
  - (#PCDATA) : parseable character Data
    <!ELEMENT nom (#PCDATA) >
  - EMPTY
    <!ELEMENT photo EMPTY >
  - ANY
    <!ELEMENT description ANY >
  - Séquence : (child1,child2,child3,…)
    <!ELEMENT étudiant (nom, prénom, tél, email) >

Les DTDs: Déclaration des éléments

- Dans une séquence, le nombre d'occurrences d'un enfant peut être contrôlé:
  - * : l'enfant est optionnel (0 ou plusieurs occurrences)
  - + : l'enfant doit apparaitre au mois un fois (ou plusieurs)
  - ? : l'enfant apparait au maximum une seul fois (i.e.: 0 ou 1)
  - Par défaut : un enfant apparait une fois exactement
- La barre oblique (|)exprime le choix entre deux éléments (soit l'un soit l'autre peut apparaitre

Exemples:

<!ELEMENT étudiant (nom, prénom, tél*, email, (CV|description)?) >

<!ELEMENT liste (étudiant+) >

Les DTDs: Déclaration des attributs

- La liste des attributs d'un élément est déclarée par:

<!ATTLIST nom-élément nom-attribut type-attribut valeur-attribut >

  - Nom-élément: nom de l'élément parent
  - Type-attribut: type des valeurs possibles de l'attribut
  - valeur-attribut: contraintes sur la valeur de l'attribut

Les DTDs: Déclaration des attributs

- Type-attribut:
  - CDATA: chaîne de caractères non analysables
  - (val1|val2|val3) : attribut acceptant une liste de valeurs
  - ID: les valeurs de l'attribut sont uniques (l'attribut peut servir d'identificateur)
  - IDREF : les valeurs de l'attribut référencent les valeurs d'un autre attribut de type ID
  - IDREFS: la valeur de l'attribut est sous forme de liste d'IDREF séparés par des espaces
  - NMTOKEN: la valeur de l'attribut respecte les règles de nommage XML
  - NMTOKENS: la valeur de l'attribut est sous forme de liste de NMTOKEN séparés par des espaces

Les DTDs: Déclaration des attributs

- valeur-attribut:
  - #REQUIRD : l'attribut est obligatoire:
    - Erreur lors de la validation si l'utilisateur ne fournit pas une valeur pour cet attribut
  - #IMPLIED : l'attribut est optionnel
    - Le parseur ne signale pas d'erreur lors de la validation si l'utilisateur ne fournit pas une valeur pour cet attribut
  - "default": on fournit une valeur par défaut à l'attribut
    - Si l'utilisateur ne fournit pas une valeur pour cet attribut, le parseur y met la valeur par défaut
  - #FIXED "default": on fournit une valeur par défaut, mais Fixe!
    - L'utilisateur ne peut mettre dans cet attribut que cette valeur fixe. S'il laisse vide, le parseur y met la valeur par défaut

Les DTDs: Déclaration des entités

- Les DTD permettent de créer des raccourcis de texte: i.e: définir des noms qui remplacent un texte long:
- Exemple d'une entité en ligne:

  <!ENTITY FSDM "Faculté des science Dhar El Mahraz" >
- Utilisation de l'entité dans le balisage XML:

<message>Rendez-vous à la &FSDM; ce soir.</message>

Les DTDs: Déclaration des entités

- Pour un texte plus long, on utilise des entités externe:
- Dans la DTD en écrit:

```
<!ENTITY FSDM SYSTEM "unFichierTexte.entity" >
```

- Dans le fichier texte en met le texte à remplacer:

«Faculté des Sciences Dhar El Mahraz,…. »

- Utilisation de l'entité dans le balisage XML:

```
<message>Rendez-vous à la &FSDM; ce soir.</message>
```


Les schémas XML (XSD):

- Les Inconvénients des DTD:
  - Les DTD ont une syntaxe non-XML
  - Les DTD ne supportent pas les espaces de noms
  - Tous les éléments sont globaux dans les DTDs
  - Pas de contrôle sur le type du contenu des éléments
- Pour combler ces lacunes: le W3C à créer un langage nommé XSD (XML Schéma Definition)

Les schémas XML (XSD):

- Le langage XSD est:
  - Un standard W3C
  - Permet de définir le schéma d'un langage XML
  - C'est un langage XML, donc utilise la syntaxe XML
  - Il supporte les espaces de noms
  - On peut créer des éléments globaux et locaux
  - On peut contrôler parfaitement les types de données

XSD: Types simples ..Types Complexes

- En XSD, les éléments d'un document XML sont soit de type simple soit de type complexe
  - Un élément de type simple est un élément qui ne contient que du texte: pas d'enfants, pas d'attributs
  - Tout élément qui contient d'autres éléments ou des attributs est de type complexe

XSD: Types simples ..Types Complexes

- Dans cet exemple:
  - Les éléments: nom, prénom, age, adresse sont simples, car ils ne contiennent que du texte
  - L'élément photo est complexe: contient un attribut
  - L'élément Liste: complexe: contient d'autre attribut
  - L'élément étudiant est complexe: contient un attribut et d'autres éléments
  - Les attributs sont toujours simples

```
<?xml version="1.0" ?>
<liste>
    <étudiant code ="E1">
        <nom>Moujtahid</nom>
        <prénom>Moujidd</prénom>
        <age>25</age>
        <adresse>25, rue Najah, Fès</adresse>
        <photo source ="moujtahid.jpg" />
    </étudiant>
...
</liste>
```


XSD: créer un schéma

- Un schéma XML est un document XML
- La racine est l'élément *schema* de l'espace de nom *http://www.w3.org/2001/XMLSchema*

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- ici les déclarations des éléments, des types simples et complexes…-->
...
...

</xsd:schema>
```

XSD: déclaration d'un élément

- La déclaration d'un élément précise son nom et son type:

```
<xsd:element name="LeNom" type="LeType" />
```

Exemple:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- ici les déclarations des éléments, des types simples et complexes…-->
...
<xsd:element name="nom" type="xsd:string" />
<xsd:element name="prénom" type="xsd:string" />
<xsd:element name="date-naissance" type="xsd:date" />
<xsd:element name="moyenne" type="xsd:decimal" />
...
</xsd:schema>
```


Les types simples prédéfinis

- XSD contient un ensemble complet de types simples couvrant tous les types de données standards connus dans les autres langages:

xsd:string , xsd:integer, xsd:decimal, xsd:float, xsd:date, xsd:uri-reference, xsd:boolean, xsd:year, xsd:language, xsd:ID , xsd:IDREF, ….

- Vous pouvez créer vos propres types simple en vous basant sur un type simple existant

Les types simples personnalisés

- Pour créer un type simple personnalisé:

```
<xsd:simpleType  name=« LeNomDuType">
      <xsd:restriction base=« LeTypeDeBase">
          <!--des restrictions ici -->
           ...
           ...
      </xsd:restriction>
</xsd:simpleType>
```

Créer un type *télType* qui personnalise le type *xsd:string* en n'acceptant que les chaînes commençant par un *0* suivi de *9* chiffres

```
<xsd:simpleType  name="télType">
      <xsd:restriction base="xsd:string">
                  <xsd:pattern value="0\d{9}" />
      </xsd:restriction>
</xsd:simpleType>
```


Les types simples personnalisés

- Plusieurs types de restrictions possibles:
  - xsd:pattern: permet de spécifier que le type accepte des chaîne de caractères conformes à une expression régulière

Exemple:

```
<xsd:simpleType  name="télType" >
      <xsd:restriction base="xsd:string">
                  <xsd:pattern value="0\d{9}" />
      </xsd:restriction>
</xsd:simpleType>
```

Les types simples personnalisés

Exemple d'expressions régulières:

- \d : un chiffre, \D : tout caractère sauf un chiffre
- \s: un espace, \S : tout caractère sauf un espace,
- Ensemble: [abc] = a, b ou c
- Intervalle : [0-9] = un chiffre entre 0 et 9
- x{5} : 5 occurrences du caractère x
- x {5,}: au minimum 5 occurrences de x
- x {5,8}: au minimum 5 occurrences de x et 8 au maximum
- x* équivaut à : {0,}
- x+ équivaut à : {1,}
- x? équivaut à : {0,1}
- (chaîne1 | chaîne2) : chaîne 1 ou chaîne 2

Les types simples personnalisés

- Plusieurs types de restrictions possibles:
  - xsd:ennumeration: permet de spécifier une liste de valeurs possible

  Exemple: la mention d'un étudiant (P, AB, B, ou TB)

```
<xsd:simpleType  name= "mentionType" >
   <xsd:restriction base="xsd:string">
       <xsd:ennumeration value="P" />
       <xsd:ennumeration value="AB" />
       <xsd:ennumeration value="B" />
       <xsd:ennumeration value="TB" />
   </xsd:restriction>
</xsd:simpleType>
```

- Dans cet exemple, on peut obtenir un résultat équivalent par:

```
<xsd:restriction base="xsd:string">
       <xsd:pattern value="P|AB|B|TB" />
</xsd:restriction>
```

Les types simples personnalisés

- Plusieurs types de restrictions possibles:
  - xsd:minInclusive: spécifie une valeur minimale (incluse)
  - xsd:maxInclusive: spécifie une valeur maximale (incluse)
  - xsd:minExclusive: spécifie une valeur minimale (exclue)
  - xsd:maxExclusive: spécifie une valeur maximale (exclue)

Exemple: la note d'un étudiant doit être entre 0 et 20:

```
<xsd:simpleType  name= "noteType" >
   <xsd:restriction base="xsd:decimal">
       <xsd:minInclusive value="0" />
       <xsd:maxInclusive value="20" />
   </xsd:restriction>
</xsd:simpleType>
```


Les types simples personnalisés

- Plusieurs types de restrictions possibles:
  - xsd:length: permet de spécifier la longueur d'une chaîne
    - Exemple: le code d'un étudiant doit être de 6 chiffres
  - xsd:minLength: spécifie la longueur minimale d'une chaîne
    - Exemple: le mot de passe doit être de 8 caractères minimum
  - xsd:maxLength: spécifie la longueur maximale d'une chaîne
    - Exemple: Un commentaire ne doit pas dépasser 500 caractères

Exemple: le code est de type ID et comporte 6 caractères:

```
<xsd:simpleType  name= « codeType" >
   <xsd:restriction base="xsd:ID">
       <xsd:length value="6" />
   </xsd:restriction>
</xsd:simpleType>
```

Les types simples personnalisés

- Plusieurs types de restrictions possibles:
  - xsd:totalDigits: le nombre total de chiffres d'un nombre décimal
  - xsd:fractionDigits: le nombre de chiffres après la virgule

Exemple: le note d'un étudiants comporte 2 chiffres après la virgules, 4 chiffres au total (14.33 est valide, 14,333 non valide)

```
<xsd:simpleType  name= "noteType" >
   <xsd:restriction base="xsd:decimal">
       <xsd:totalDigits value="4" />
        <xsd:fractionDigits value="2" />
   </xsd:restriction>
</xsd:simpleType>
```


Les types complexes

- Un élément complexe contient d'autre éléments ou des attributs
- Tandis que les types simples décrivent le contenu d'un élément simple, les types complexes décrivent la structure d'un élément complexe

Les types complexes

- **Exemple: l'élément liste est un élément complexe. Il contient un ou plusieurs occurrences de l'élément étudiant et un attribut année obligatoire avec valeur fixe.**

```
<?xml version="1.0">
<liste année="2018">
   <étudiant  code="E1">
       <nom>Moujtahid</nom>
       <prénom>Moujidd</prénom>
   </étudiant>
   ...
   ...
</liste>
```


Les types complexes

- **Exemple: l'élément liste est un élément complexe. Il contient un ou plusieurs occurrences de l'élément étudiant. On le déclare comme suit:**

```
<xsd:element name="liste" type="listeType">

<xsd:complexType name="listeType">
   <xsd:sequence>
     <xsd:element name="étudiant" type="EtudiantType" minOccurs="0"
maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:attribute name="année" type="xsd:year" use="required" fixed="2018" />

</xsd:complexType>
```

Les types complexes

- Un type complexe est déclaré par la balise <xsd:complexType>
- On définit la séquence d'éléments enfants
- On contrôle le nombre d'occurrences d'un enfant par minOccurs et maxOccurs
- Après la fermeture de la balise séquence, on définit la liste des attributs en répétant l'élément xsd:attribute.
- Un attribut possède:
  - Name (le nom de l'attribut)
  - type (un type simple)
  - use = (required, implied, prohibited)
  - fixed , default (éventuellement)

Les types complexes: Eléments locaux et globaux

- Dans cette exemple: l'élément étudiant est définit (name) à l'intérieur d'un type complexe. On dit qu'il est local. Il n'est pas visible depuis l'extérieur du type:

```
<xsd:element name="liste" type="listeType">

<xsd:complexType name="listeType">
   <xsd:sequence>
     <xsd:element name="étudiant" type="EtudiantType" minOccurs="0"
maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:attribute name="année" type="xsd:year" use="required" fixed="2018" />

</xsd:complexType>
```

Les types complexes: Eléments locaux et globaux

- Dans cette exemple: l'élément étudiant est définit sous la racine directement. On dit qu'il est global.
- Un élément global peut être référencé (ref) dans tous type complexe, à chaque fois qu'on a besoin

```
<xsd:element name="étudiant" type="EtudiantType" />
<xsd:element name="liste" type="listeType"  />

<xsd:complexType name="listeType">
   <xsd:sequence>
     <xsd:element ref="étudiant" minOccurs="0" maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:attribute name="année" type="xsd:year" use="required" fixed="2018" />
</xsd:complexType>
```


Les feuilles de style

Cette partie concerne l'affichage du contenu d'un document XML:

- **Les feuilles de style**
  - CSS
  - XSL (XML StyleSheet Language)

Feuilles de style XML

- Comment afficher un document XML ?
  - En HTML, tout le monde sait comment va s'afficher le texte suivant:

    <h1> affichage en HTML </h1>

  - Mais, comment va s'afficher l'élément XML suivant:

    <xy1e3> affichage en HTML </xy1e3>

Feuilles de style XML

- HTML :
  - les navigateurs connaissent déjà comment afficher et présenter les balises HTML
  - Les balises HTML possèdent des styles sous-jacent
- XML :
  - il est logique que les navigateurs ne connaissent rien à propos du style des nouveaux éléments crées par les utilisateurs
  - Pour afficher du XML: Il faut définir le style

Feuilles de style XML

- Les feuilles de style en cascades(CSS) :
  - Utilisée avec HTML
  - Permettent de dissocier le style du balisage HTML
  - Peuvent être utilisées avec XML, et de la même manière

```
h1 {color: red ; text-decoration: underline}
p {margin: 1cm; font-size: 14 pt;}
```

CSS avec HTML

```
email {margin:1cm; padding:1cm;
border:2pt; border-color:red ;
background-color: yellow;border-
style:solid; display:block}
date {display:block}
from {display:block}
to {display:block}
subject {display:block; color:red}
```

CSS avec XML

Feuilles de style XML

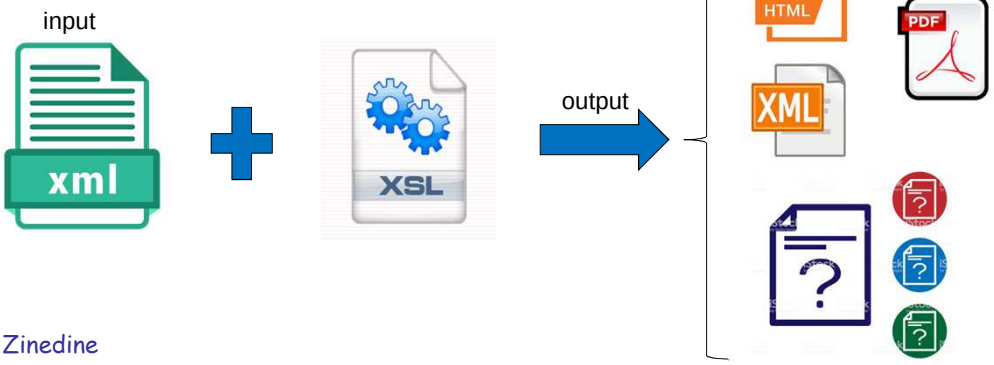
- Deux façons pour définir les styles:
  - CSS (Cascading Style Sheet)
  - XSL (XML StyleSheet Language)

Feuilles de style XML

- Les CSS n'ont pas une syntaxe XML
- Leur rôle est trop limité
- Le W3C a créé un langage très puissant qui peut faire le rôle des CSS et bien beaucoup d'autres choses: XSL
- XSL : XML Stylesheet Language
- XSL comporte deux composantes: XSLT et XSL-FO
- XSLT (XSL Transform) : permet de transformer un document XML en un autre format
- XSL-FO: formating Objects (donne l'équivalent XML des CSS)

Feuilles de style XML: XSLT

- XSLT: XSL Transform : permet de transformer un document XML en un autre format
- Principe:



Feuilles de style XML

- Une feuille de style XSLT est un document XML
- La racine est `<xsl:stylesheet>` qui appartient à l'espace de nom `http://www.w3.org/1999/XSL/Transform.`
- Exemple :

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


Feuilles de style XML

- Une feuille de style contient un ou plusieurs templates chacun applicable à un jeu de nœuds défini par une expression Xpath
- Seul le template racine est obligatoire (correspond à la racine /)

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

Feuilles de style XML

- **Pour illustrer, nous allons utiliser le document XML suivant:**

```
<?xml version = "1.0" ?>
<?xml-stylesheet type= "text/xsl"
href = "liste.xsl" ?>
<liste année = "2019">
   <étudiant  code="E1">
       <nom>Moujtahid</nom>
       <prénom>Moujidd</prénom>
       </étudiant>
   <étudiant code="E2">
              <nom>Kaddouri</nom>

       <prénom>Kaddour</prénom>
              <age>26</age>
   </étudiant>
   ...
   ...
</liste>
```

affichage

Liste des étudiants réussis

Code	Nom	Prénom	Note	Mention
E1	Moujtahid	Moujidd	17	TB
E2	Kaddouri	Kaddour	14.5	B
E3	Jallouli	Jalloul	13	AB
E4	Kaslani	Kassoul	3.5	Aj

89

Feuilles de style XML

- **Pour arriver à cet affichage, nous devons transformer le document XML en le document HTML Suivant:**

```
<html>
<body>
<h1 align="center" >Liste des étudiants réussis</h1>
<table border="1" align="center" bordercolor="red" cellpadding="10">
<tr>
<th>Code</th><th>Nom</th><th>Prénom</th><th>Note</th><th>Mention</th></tr>
<tr> <td>E1</td><td>Moujtahid</td><td>Moujidd</td><td>17</td><td>TB</td></tr>
<tr> <td>E2</td><td>Kaddouri</td><td>Kaddour</td><td>14.5</td><td>B</td></tr>
<tr> <td>E3</td><td>Jallouli</td><td>Jalloul</td><td>13</td><td>AB</td></tr>
<tr> <td>E4</td><td>Kaslani</td><td>Kassoul</td><td>3.5</td><td>Aj</td></tr>
</table>
</body>
</html>
```

A. Zinedine

90

Ajouter du texte à la sortie

- Dans un template, tout texte non-XSL sera généré tel quel
- Cet exemple génère dans le output des balises HTML et un titre

```
<?xml version = "1.0" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<html><body>
<h1 align="center" >Liste des étudiants réussis</h1>
</body></html>
</xsl:template>
</xsl:stylesheet>
```


Boucler avec For-each

- On utilise For-each pour extraire un jeu de nœud et appliquer sur tous ses éléments un traitement.
- Exemple: On extrait et on parcourt tout les nœuds étudiants. À chaque nœud trouvé, on affiche Bonjour:

```
<?xml version = "1.0" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<html><body>
<h1 align="center" >Liste des étudiants réussis</h1>
<xsl:for-each select="/liste/étudiant">
   Bonjour
</xsl:for-each>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

Extraire la valeur d'un nœud par Value-of:

- On extrait la valeur d'un élément en utilisant value-of
- L'exemple suivant affiche les noms de tous les étudiants
- N.B: « nom » n'est pas précédé par « / », donc c'est relatif à l'élément en cours

```
<?xml version = "1.0" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<html><body>
<h1 align="center" >Liste des étudiants réussis</h1>
<xsl:for-each select="/liste/étudiant">
   <xsl:value-of select="nom" />
</xsl:for-each>
</body></html>
</xsl:template>
</xsl:stylesheet>
```


Extraire la valeur d'un nœud par Value-of:

- Améliorer l'affichage par une table:

```
...
<xsl:template match="/">
<html><body>
<h1 align="center" >Liste des étudiants réussis</h1>
<table border="1" align="center" bordercolor="red" cellpadding="10">
<tr><th>Code</th><th>Nom</th><th>Prénom</th><th>Note</th><th>Mention</th></tr>
<xsl:for-each select="/liste/étudiant">
   <tr>
     <td><xsl:value-of select="code" /></td>
     <td><xsl:value-of select="nom" /></td>
     <td><xsl:value-of select="prénom" /></td>
     <td><xsl:value-of select="note" /></td>
   </tr>
</xsl:for-each>
</table>
...
```

Ordonner le jeu du résultat par sort:

- La balise xsl:sort doit suivre xsl:for-each immédiatement :

```
...
<xsl:template match="/">
<html><body>
<h1 align="center" >Liste des étudiants réussis</h1>
<table border="1" align="center" bordercolor="red" cellpadding="10">
<tr><th>Code</th><th>Nom</th><th>Prénom</th><th>Note</th><th>Mention</th></tr>
<xsl:for-each select="/liste/étudiant">
<xsl:sort select="nom" data-type="text" order="ascending" />
   <tr>
     <td><xsl:value-of select="code" /></td>
     <td><xsl:value-of select="nom" /></td>
     <td><xsl:value-of select="prénom" /></td>
     <td><xsl:value-of select="note" /></td>
   </tr>
</xsl:for-each>
</table>
...
```

Faire des tests avec xsl:if

- xsl:if permet de faire des transformations conditionnées
- Dans cet exemple, on n'affiche que les étudiants dans la note est supérieure à 10

```
<xsl:template match="/">
...
<xsl:for-each select="/liste/étudiant">
<xsl:sort select="nom" data-type="text" order="ascending" />
<xsl:if test="note &gt; 10">
   <tr>
     <td><xsl:value-of select="code" /></td>
     <td><xsl:value-of select="nom" /></td>
     <td><xsl:value-of select="prénom" /></td>
     <td><xsl:value-of select="note" /></td>
   </tr>
<xsl:if>
</xsl:for-each>
...
```

Faire plusieurs tests avec xsl:choose:

- on affiche la mention de chaque étudiant à base de sa note:

```
...
<xsl:for-each select="/liste/étudiant">
<xsl:sort select="nom" data-type="text" order="ascending" />
<xsl:if test="note &gt; 10">
<tr><td><xsl:value-of select="code" /></td>
     <td><xsl:value-of select="nom" /></td>
     <td><xsl:value-of select="prénom" /></td>
     <td><xsl:value-of select="note" /></td>
     <td>
        <xsl:choose>
           <xsl:when test="note &gt; 10 and note &lt; 12"> P </xsl:when>
           <xsl:when test="note &gt; 12 and note &lt; 14"> AB </xsl:when>
           <xsl:when test="note &gt; 14 and note &lt; 16"> B </xsl:when>
           <xsl:when test="note &gt; 16"> P </xsl:when>
        </xsl:choose" />
    </td></tr>
<xsl:if>
</xsl:for-each>
```

XQuery

- XQuery est un langage de requête pour les documents XML
- XQuery pour XML est comme SQL pour les BD relationnelles
- XQuery se base principalement sur les expressions Xpath
- XQuery est une recommandation W3C

XQuery: règles de syntaxe de base

- XQuery est sensible à la casse
- Les éléments, attributs, et variables déclarés dans XQuery doivent être des noms XML valides
- Toute chaîne de caractères dans XQuery doit être entre doubles ou simples guillemets
- Toute variable dans XQuery doit commencer par un $
- Les commentaires XQuery sont délimités par (: et :)

Comment sélectionner les nœuds d'un document

- Fonctions : doc("liste.xml")
- Expressions XPath : doc("liste.xml")/liste/étudiant
- Prédicats : doc("liste.xml")/liste/étudiant ["note > 10"]

Sélectionner des nœuds avec FLWOR

- FLWOR = For..Let..Where..Order by..Return
- Exemple:

```
for $x in doc("liste.xml")/liste/étudiant
let $i := 10
where $x/note >= $i
order by $x/note descending
return data($x/nom)
```

- For: permet de créer une variable $x, charger *liste.xml*, puis récupérer un jeu de nœud pour le stocker dans $x
- Let : permet de déclarer davantage de variable
- Where : on ne garde que les nœud qui répondent à cette condition
- Order by: ordonne le résultat selon un critère
- Return: les données à afficher

Formater le résultat avec HTML

- Exemple:

```
(: Exemple Xquery :)

<html>
<body>
<h1>Liste des étudiants plus agés que Kaddouri </h1>
<hr />
<ol>
{
   for $x in doc("liste.xml")/liste/étudiant
   let $i := 10
   where $x/note >= $i
   order by $x/note descending
   return <li> {data($x/nom)} </li>
}
</ol>
</body>
</html>
```

Expressions conditionnelles de XQuery

- If…then…else:

```
(: Exemple Xquery :)
<html>
<body>
<h1>Liste des étudiants plus agés que Kaddouri </h1>
<hr />
<ol>
{  for $x in doc("liste.xml")/liste/étudiant
   return
<li> {data($x/nom)}(
   if ($x/note &gt; 10 ) then 'Admis'
   else 'Ajourné'  )
 </li>
}
</ol>
</body>
</html>
```

XPath

- XPath est un élément majeur dans la technologie XML
- XPath permet de naviguer entre les nœuds d'un document XML
- Xpath permet d'exprimer d'une manière exacte les jeux de nœuds à extraire pour traitement
- Plusieurs langages reposent sur Xpath tels que XSLT et XQuery

Expressions XPath

- La notation utilisée par XPath ressemble à celle utilisée par les systèmes d'exploitation
- La racine: /
- Le nœud courant: .
- L'enfant d'un nœud: parent/enfant/petit-enfant
- Le parent: ..
- Le descendant: parent/*/petit-enfant
- Tous les descendants : parent//
- Tous les éléments *nom* sous l'élément *liste*: /liste//nom
- Les attributs: @nom-de-l'attribut
- Les prédicats: /liste/étudiant[note > 10]

Fonctions XPath

- XPath offre un ensemble de "fonctions" :
  - Position(): donne la position d'un nœud
    - Afficher le nom du 3ème étudiant:

```
<xsl:value-of select="/liste/étudiant[position()=3]/nom" />
```

  - Last(): donne la position du dernier nœud d'un jeu
    - Afficher le nom du 3ème étudiant:

```
<xsl:value-of select="/liste/étudiant[position()=last()]/nom" />
```

  - Sum(): return la somme des valeurs d'un jeu de nœuds
  - Count(): return le nombre des nœuds dans un jeu
    - Afficher la moyenne de la classe:

```
<xsl:value-of
select="sum(/liste/étudiant/note) div count(/liste/étudiant)" />
```

Fonctions XPath

- XPath offre un ensemble de "fonctions" :
  - Format-number: permet de formatter un nombre pour affichage : plusieurs formats offerts
  - Floor(x) : la partie entière de x
  - Ceiling(x) = Floor(x)+1
  - Round(x): l'entier le plus proche à x (soit =floor soit =ceiling)

Fonctions XPath

- XPath offre un ensemble de "fonctions" :
  - Substring(S,n,m) : extrait m caractère à partir de la position n de la chaîne S
  - Substring-before(S,s): recherche la sous-chaine s de S et extrait tout ce qui est avant la première occurrence
  - Substring-after(S,s): recherche la sous-chaine s de S et extrait tout ce qui est après la première occurrence
  - Translate (S,S1,S2') : prend la chaine S en entrée et remplace tous ses caractères qui figurent dans S1 par les caractères correspondant en position dans la chaine S2

Autres Standards

- Un très grand nombre de standards:
- XLink
- XPointer
- MathML
- SVG
- RDF
- …
- …

Parseurs

- XML est uniquement un langage de structuration et de représentation de données
- Il ne comporte pas d'instructions de contrôle et ne permet donc pas d'exploiter directement les données
- Pour réaliser des applications XML, il faut donc avoir recours aux parseurs.
- Toute applications XML passe par une phase préalable d'analyse du document XML par un parseur.

Parseurs

- Un parseur XML (ou analyseur syntaxique), permet de récupérer dans une structure XML, des balises, leur contenus, leurs attributs et de les rendre accessibles.
- Il existe deux types de parseurs :
  - Les parseurs SAX (Simple API for XML), orientés événement
  - Les parseurs DOM (Document Object Model) orientés hiérarchie

Parseurs SAX

- SAX fournit une interface événementielle pour parcourir un document XML
- Cette API renvoie aux applications des "événements" (ouverture de balise, fermeture de balise, contenu textuel...)
- Elle permet donc de traiter à la volée l'occurrence de telle ou telle balise.

Parseurs SAX

- SAX est bien adapté pour :
  - les traitements qui ne nécessitent qu'une seule passe sur le document
  - le cas de gros volumes de données
  - Les cas où il n'est pas nécessaire d'avoir une représentation complète des données en mémoire.

Parseurs DOM

- Le modèle DOM est une recommandation du W3C
- Les parseurs DOM, à la différence des parseurs SAX, utilisent une approche hiérarchique.
- Ce sont les plus souvent rencontrés.

Parseurs DOM

- Ils permettent une navigation aisée dans un document
- nécessitent le chargement complet en mémoire de la structure arborescente du document XML.
- On stocke ainsi un arbre DOM.

Parseurs DOM

- Un parseur DOM prend en entrée un document XML et construit, à partir de cela, un arbre formé d'objets
- chaque objet appartient à une sous-classe de l'objet Node
- Des opérations sur ces objets permettent de créer de nouveaux nœuds, ou de naviguer dans le document.

L'API DOM

- L'API DOM décrit un ensemble de méthodes et classes permettant de naviguer et d'éditer un document XML
- Les spécifications du W3C ne proposent pas d'implémentation: libre à chaque éditeur et langage de proposer sa propre bibliothèque
- La plupart des langages ont un implémentation du DOM
- L'API DOM est indépendante de toute plate-forme ou langage de programmation

L'API DOM

Dans la suite, nous présentons l'essentiel de cette API

- Node
- NodeList
- NamedNodeMap
- DOMImplimentation
- DOMException

- Document (9)
- ProcessingInstruction (7)
- DocumentType (10)
- DocumentFragment (11)
- Element (1)
- Attr (2)
- CaracterData
  - Text (3)
  - Comment (8)
  - CDATASection (4)
- Entity (6)
- EntityReference (5)
- Notation (12)

Node

- nodeName
- NodeType
- nodeValue
- parentNode
- childNodes
- firstChild
- lastChild
- previousSibling
- nextSibling
- attributes

L'objet Document

- documentElement (racine)
- docType

Document

- CreateAttribute(name) [Attr Obj]
- CreateCDATASection(data)
- createComment(data)
- createElement(name)
- createEntityReference(name)
- createProcessingInstruction(target,data)
- createTextNode(data)
- createDocumentFragment() (empty)

Node

- appendChild(child)
- insertBefore(child,before)
- replaceChild(child,toReplace)
- removeChild(child)
- cloneNode(deep)
- hasChildNodes()

Element

- setAttribute(name,value)
- getAttribute(name)
- removeAttribute(name)

- setAttribute(Attr) [Attr Obj]
- setAttributeNode(attr)
- getAttributeNode(attr)
- removeAttributeNode(node)

CaracterData

- appendData(data)
- insertData(offset,data)
- deleteData(offset,length)
- replaceData(offset,length,data)
- substringData(offset,length)

Fin ☺