Université Ahmed Zabaneh - Relizane
Faculté des Sciences et de la Technologie
Département d'Informatique

Données Semi Structurées

Chapitre 2: Noyau XML
Partie 2

S. Bella
Bella.salyma@gmail.com

L3 – Systèmes Informatiques

2021/2022

Plan

Espace de noms (Name Spaces)

Espace de noms

Résoudre les conflits d'utilisation des balises

- La société 4 **intègre** au document 4 des documents proviennent des sociétés 1, 2 et 3.
- Chaque document définit ses **propres** balises et attributs.
- **Problème**: la balise de température dans document (1, 2, 3) est une température de la salle, du corp humain, ou du processeur ?.
- **Solution**: classer chaque concept (la porté) dans une classe (espace des noms), pour savoir comment il doit être **interprété**.

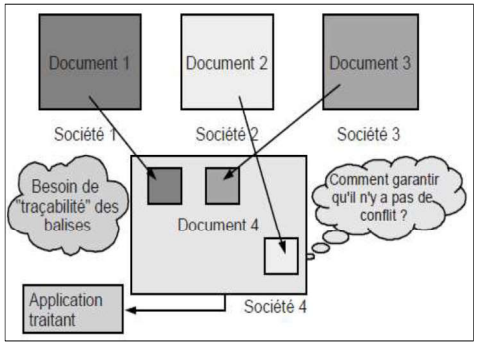


Figure 1.2. Croisement des documents

Espace de noms

- Les espaces de noms sont un concept très commun en informatique, ont été introduits en XML afin de pouvoir **mélanger plusieurs vocabulaires** au sein d'un même document.
- Par analogie, cette notion est reprise dans la plupart des langages de programmation. Par exemple, dans le langage de programmation Java, les packages servent à **délimiter la portée** d'une classe.
- L'utilisation des espaces de noms garantit une forme de **traçabilité** de la balise (l'origine de la balise…) et évite les **ambiguïtés** d'usage.
- Pour que les espaces de noms aient un sens, il faut pour chacun d'eux un **identifiant unique (URI)**.

URI (Uniform Resource Identifier)

- La terminologie XML distingue les **URI** (Uniform Resource Identifier) et les **URN** (Uniform Resource Name). Les **URL** (Uniform Resource Locator) sont bien connues car elles sont utilisées au quotidien pour naviguer sur le WEB.
- La notion la plus générale est celle d'URI; les URI comprennent les URL (Uniform Resource Locator) et les URN (Uniform Resource Name) même si certains URI peuvent être simultanément des URL et des URN.

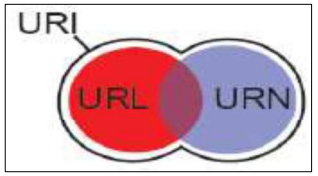
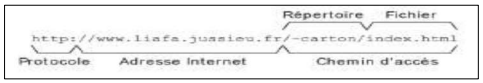


Figure 2.2. Relation entre URI, URL et URN

URI (Uniform Resource Identifier)

- Un URI est un **identifiant** qui permet de désigner **sans ambiguïté** un document ou plus généralement une ressource. Cet identifiant doit donc être **unique** de manière **universelle**.
- URL est un **localisateur** qui identifie un document en spécifiant un mécanisme pour le retrouver. Elle est composée d'un **protocole** suivi d'une **adresse** permettant de récupérer le document avec le protocole. Par exemple :



- URN est un **nom** donné à un document indépendamment de la façon d'accéder au document. Un exemple typique d'URN est l'URN formé à partir du numéro ISBN d'un livre comme urn:isbn:978-2-7117-2077-4. Cet URN identifie le livre Langages formels, calculabilité et complexité mais n'indique pas comment l'obtenir.

Identification d'un espace de noms

- Un espace de noms est identifié par un URI appelé **URI de l'espace de noms**.
- Cet URI garantit seulement que l'espace de noms soit identifié de manière **unique**. Exemple :

  <ns1:titre> la balise titre appartient à l'espace de nom ns1.

  <ns2:titre> la balise titre appartient à l'espace de nom ns2.

  <ns1:titre> et **<ns2:titre>** sont deux balises bien différentes.

Déclaration d'un espace de noms

- Un espace de noms est déclaré par un **pseudo-attribut** dont le nom prend la forme **xmlns:prefix** (ns: namespace) où **prefix** est un nom XML ne contenant pas le caractère '**:**'. La valeur de ce pseudo-attribut est **l'URI** qui identifie l'espace de noms, et le préfixe est ensuite utilisé pour ***qualifier* les noms d'éléments.**

- Un ***nom qualifié*** d'élément prend la forme ***prefix*:*local*** où ***prefix*** est un préfixe associé à un espace de noms et *local* est le *nom local* de l'élément. Ce nom local est également un nom XML ne contenant pas le caractère **':'.**

- Il existe deux méthodes pour intégrer des espaces de noms: **Implicite, Explicites** (souvent employée).

Espace de nom implicite (par défaut)

- Préciser par un pseudo-attribut **xmlns** (préfixe vide) dont la valeur est l'URI de l'espace de noms.
- **L'espace de noms par défaut** s'applique à l'élément où se situe sa déclaration et à tout son contenu.

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

URI (une adresse fictive qui n'as aucun existence)

- Ici, l'élément *chapitre* est dans l'espace http://www.masociete.com. C'est également le cas de l'élément *paragraphe*, puisqu'il est dans l'élément *chapitre*.
- Un espace de nom **par défaut ne concerne que les éléments**. Les attributs et les textes n'y appartiennent pas. Le texte n'est jamais dans un espace de nom puisqu'il représenté les données.

Espace de nom implicite (par défaut)

- Les espaces de noms ne sont pas **imbriqués**; on ne peut appliquer qu'un seul espace de noms à la fois.

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

- L'élément *paragraphe* n'appartient pas à l'espace de noms http://www.masociete.com mais uniquement à l'espace de noms http://www.autresociete.com.
- L'espace de nom par défaut présente l'inconvénient d'être **peu contrôlable** sur un document de taille importante. En effet, tout ajout ou modification d'un tel espace va se répercuter sur la totalité du contenu.

Espace de nom explicite

- La syntaxe de cet espace introduit la notion de **préfixe**.
  - Un préfixe est une sorte **de raccourci vers l'URL de l'espace de noms**.
  - On déclare un préfixe comme un pseudo-attribut commençant par **xmlns:prefixe**.
  - Une fois déclaré, il est employable uniquement dans l'élément le déclarant et dans son contenu.
  - L'emploi consiste à ajouter en **tête** de l'élément, de **l'attribut** ou d'une **valeur** d'attribut, le préfixe suivi de '**:**'.

- Exemple:

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

- L'élément *resultat* est dans l'espace de noms http://www.masociete.com grâce au préfixe p.

Espace de nom explicite

- **Combination des espaces**: on peut déclarer et utiliser plusieurs espaces de noms grâce aux préfixes. Exemple :

```
<p1:res xmlns:p1="http://www.masociete.com" xmlns:p2="http://www.autresociete.com" >
  <p2:res>
  </p2:res>
</p1:res>
```

- Le premier élément *res* est dans l'espace de noms http://www.masociete.com alors que l'élément *res* à l'intérieur est dans l'espace de noms http://www.autresociete.com.

- Les espaces de nom peuvent s'appliquer via un préfixe sur un **attribut** ou une **valeur** d'attribut.

Exemple:

```
<livre xmlns:p="http://www.imprimeur.com" p:quantite="p:50lots">
    <papier type="p:A4"/>
</livre>
```

- L'attribut *quantité* et les valeurs d'attribut *50lots* et *A4* sont dans http://www.imprimeur.com.

Espace de noms XML

- Le préfixe **xml** est toujours implicitement lié à l'espace de noms XML identifié par l'URI **http://www.w3.org/XML/1998/namespace**. Cet espace de noms n'a pas besoin d'être déclaré.

- Les quatre attributs particuliers **xml:lang, xml:space, xml:base** et **xml:id** font partie de cet espace de noms. Ces quatre attributs sont déclarés par le schéma XML qui se trouve à l'URL **http://www.w3.org/2001/xml.xsd**

Quelques espaces de noms classique

- **XML** http://www.w3.org/XML/1998/namespace
- **Xinclude** http://www.w3.org/2001/XInclude
- **Xlink** http://www.w3.org/1999/xlink
- **MathML** http://www.w3.org/1998/Math/MathML
- **XHTML** http://www.w3.org/1999/xhtml
- **SVG** http://www.w3.org/2000/svg
- **Schémas** http://www.w3.org/2001/XMLSchema
- **Instances de schémas** http://www.w3.org/2001/XMLSchema-instance
- **Schematron** http://purl.oclc.org/dsdl/schematron
- **XSLT** http://www.w3.org/1999/XSL/Transform
- **XSL-FO** http://www.w3.org/1999/XSL/Format
- **DocBook** http://docbook.org/ns/docbook
- **Dublin Core** http://purl.org/dc/elements/1.1/

Exercice 1 : Ce document utilise le préfixe *fact*. Est-il correct ? Corrigez-le :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fact:facture>
<fact:montant> 10$ </fact:montant>
<fact:nom> Jean </fact:nom>
</fact:facture>
```

Solution 1 : N'est pas correcte, il faut déclarer l'espace de nom :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fact:facture xmlns:fact= "www.facture-dz.come">
<fact:montant> 10$ </fact:montant>
<fact:nom> Jean </fact:nom>
</fact:facture>
```

Exercice 2: On considère le document suivant :

```
<?xml version="1.0"?>
<recherche
  xmlns="http://www.google.com/"
  xmlns:yahoo="http://www.yahoo.com/"
  xmlns:twitter="http://www.twitter.com/">
<mots-clefs xmlns="http://www.yahoo.com/"> fraise, orange </mots-clefs>
<yahoo:mot xmlns:yahoo2="http://www.yahoo.com/">
 <nombre>3</nombre>
</yahoo:mot>
<twitter:mot xmlns:twitter2="http://www.twitter.com/">
 <nbre xmlns="">4</nbre>
</twitter:mot>
</recherche>
```

a. À quel espace de noms l'élément *mots-clefs* appartient?
b. À quel espace de noms l'élément *recherche* appartient?
c. À quel espace de noms l'élément *yahoo:mot* appartient?
d. À quel espace de noms l'élément *nombre* appartient?
e. À quel espace de noms l'élément *nbre* appartient?
f. Combien y a-t-il d'espaces de noms dans ce document?

Validation d'un document XML (DTD/Schémas XML)

Validation d'un document XML

Le contenus d'un document XML provient de plusieurs ressources :

- Echange de donnée (réseau, logiciels, machines,…),
- Opération manuelle (Saisie).
- Saisie automatique (Exportation à partir d'une BDD ou d'autre document).

Problème: une erreur (de saisie ou de programmation) engendre un mal fonctionnement du système.

Solution: La validation a pour objectif de **garantir** qu'un document XML est bien structuré sans aucune incohérence, donc :

- La validation renforce la **qualité des échanges** en contraignant l'émetteur de données et le consommateur de données à vérifier la cohérence des données structurées en XML.
- Par cohérence, il faut entendre à la fois le **vocabulaire** (éléments, attributs et espaces de noms) mais également, chose aussi importante, **l'ordre et les quantités**.

→ Donc, un document valide est un document **bien formé** (correcte syntaxiquement) et **conforme à une grammaire** DTD ou au schéma XML.

DTD: Document Type Definition

- Une DTD (Document Type Definition) est une forme de **grammaire** (ensemble de règles) utilisée pour **définit la structure** d'un document XML, ses éléments, ses attributs et ses entités.
- DTD fournit des règles que le parseur doit suivre pour la **validation correcte** du document XML.
- La syntaxe d'une DTD n'est pas à base de balise mais à **base d'instructions**, selon la syntaxe **<!INSTRUCTION…>.**
- La déclaration de la DTD du document doit être **placée dans le prologue**. Cette déclaration définit : le nom des éléments, leur contenu, le nombre de fois et l'ordre d'apparition; les attributs éventuels et leurs valeurs par défaut; les noms des entités qui peuvent être utilisées,…

Type de DTD

La DTD est définie par la balise: **<!DOCTYPE..>**

- **Interne** au document XML: **<!DOCTYPE *element-racine* [*declarations*]>**

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE parent [
<!ELEMENT parent (garcon,fille)>
<!ELEMENT garcon (#PCDATA)>
 <!ELEMENT fille (#PCDATA)>
 ]>
 <parent>
    <garcon>Samir</garcon>
    <fille>Souad</fille>
 </parent>
```


Type de DTD

La DTD est définie par la balise: **<!DOCTYPE..>**

- Dans un fichier **Externe** (son extension .dtd):**<!DOCTYPE *element-racine SYSTEM/Public "url"*>**

  - Privée utilisant le mot clé ***System*** (fichier accessible uniquement en local):

    <!DOCTYPE etudiants SYSTEM "etudiants.dtd">

  - Publique utilisant le mot clé ***Public*** (disponible pour tous le monde):

    <!DOCTYPE etudiants Public "http://www...../etudiants.dtd">

- Le chemin ***url*** vers la DTD externe peut être soit une URL complète commençant par http:// ou ftp:// soit plus simplement le nom **d'un fichier local** (se trouve dans le dossier que le fichier xml).

Type de DTD

La DTD est définie par la balise: **<!DOCTYPE..>**

- **Mixte**: Il est possible d'avoir simultanément une DTD externe et une DTD interne. La DTD globale est alors formée des déclarations internes suivies des déclarations externes.

- Il n'est pas possible de déclarer plusieurs fois le même élément dans une DTD. Lorsque la DTD est mixte, tout élément doit être déclaré dans la partie interne ou dans la partie externe mais pas dans les deux.

Contenu d'une DTD

- Une DTD peut contenir:
  - Des déclarations d'éléments,
  - Des déclarations de listes d'attributs,
  - Des déclarations d'entités,

- Chacune de ces déclarations commence par la chaîne suivi d'un mot clé qui indique le type de déclaration '<!'.

- Les mots clés possibles sont **ELEMENT**, **ATTLIST** et **ENTITY**.

- La déclaration se termine par le caractère '>'.

Déclaration d'élément

- Syntaxe d'un élément (balise en XML): **<!ELEMENT *nom_elt contenu*>**
- ***contenu*** indique le type de données ou des éléments qui doivent ou peuvent être imbriqués.
- Le ***contenu*** peut être un :
  - *Texte* → **#PCDATA** (Parseable Character Data: données caractères parsées): correspond au texte contenu entre les balises de début et de fin de l'élément.

```
<!ELEMENT Nom (#PCDATA)>        ---DTD
<Nom>Mahmoudi med</Nom>         ---XML
```

  - *Vide* → **EMPTY:** l'élément n'a pas de contenu mais il est possible d'avoir des attributs.

```
<!ELEMENT image EMPTY>          ---DTD
<image/>                        ---XML
```

  - *Quelconque* → **ANY** : l'élément peut contenir n'importe quel type de donnée.

```
<!ELEMENT adresse ANY>          ---DTD
```


Déclaration d'élément

- Le ***contenu*** peut être un :
  - ***Séquence d'élément*** : suite de noms d'éléments **séparés** par des **virgules**. L'ordre doit être respecté.

```
<!ELEMENT personne (prenom, nom)>   ---DTD
<personne>                          ---XML
   <prenom>(...)</prenom>
   <nom>(...)</nom>
</personne>
```

  - ***Choix*** : suite de noms d'éléments **séparés** par des **barres verticales "|".**

```
<!ELEMENT personne (nom_prenom|nom)>   ---DTD
<personne>                             ---XML
   <nom_prenom>(...)</nom_prenom>
   <nom>(...)</nom>
</personne>
```


Déclaration d'élément

- Le ***contenu*** peut être un :
  - ***Mélange*** : l'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.

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

<citation> Être ou ne pas être
    <auteur>Shakespeare</auteur>          ---XML
    Être ou ne pas être
</citation>
```


Indicateurs d'occurrence

- Les contenus (élément ou groupe d'éléments) peuvent être quantifiés par les opérateurs ***,+** et **?.** Ces opérateurs sont liés au concept de cardinalité. Lorsqu'il n'y a pas d'opérateur, la quantification est de 1 (donc toujours présent).
- Voici le détail de ces opérateurs :
  - *** :** 0 à n fois (l'élément peut être présent plusieurs fois ou aucune).
  - **+ :** 1 à n fois (l'élément doit être présent au minimum une fois).
  - **? :** 0 ou 1 fois (l'élément peut être optionnellement présent).
- Exemples: **<!ELEMENT plan (introduction?,chapitre+,conclusion?)>**
  L'élément *plan* contient un élément *introduction* optionnel, suivi d'au moins un élément *chapitre* et se termine par un élément *conclusion* optionnel également.

  <!ELEMENT chapitre (auteur*,paragraphe+)>
  L'élément *chapitre* contient de 0 à n éléments *auteur* suivi d'au moins un élément *paragraphe*.

Déclaration d'attributs

- Syntaxe d'un attribut: **<!ATTLIST nom_elt nom_atrb type obligation valeur_par_defaut>**
- **type** de l'attribut peut être:
  - ***CDATA*** **:** du texte (Character Data) → **le plus utilisé**,
  - ***ID*** : un identifiant unique (combinaison de chiffres et de lettres) → **le plus utilisé**,
  - ***IDREF*** : une référence vers un ID → **le plus utilisé**,
  - **IDREFS** : une liste de références vers des ID (séparation par un blanc),
  - ***NMTOKEN*** : un mot (donc pas de blanc),
  - ***NMTOKENS*** : une liste de mots (séparation par un blanc),
  - ***Enumération de valeurs*** : chaque valeur est séparée par le caractère **|**.

Déclaration d'attributs

- **obligation**:
  - ***#REQUIRED*** : attribut obligatoire.
  - ***#IMPLIED*** : attribut optionnel.
  - ***#FIXED*** : attribut toujours présent avec une valeur.
- **valeur_par_defaut** est présente pour l'énumération ou lorsque la valeur est typée avec *#IMPLIED* ou *#FIXED*.
- **Exemples**:

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

→ L'élément ***chapitre*** possède un attribut ***titre*** **obligatoire** et un attribut ***auteur*** **optionnel**.

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

→ L'élément ***crayon*** possède un attribut ***couleur*** dont les valeurs font partie de l'ensemble ***rouge, vert, bleu*** dont la valeur par défaut est ***bleu***.

Exemple complet (DTD et XML)

```
<!DOCTYPE magasin [
<!ELEMENT magasin (service+)>
<!ELEMENT service (produit*)>
<!ATTLIST service code ID #REQUIRED>
<!ELEMENT produit (#PCDATA)>
<!ATTLIST produit code ID #REQUIRED>
]>
<magasin>
 <service code="A001">
  <produit code="DE205"> Soupe </produit>
  <produit code="TM206"> Condiment </produit>
 </service>
 <service code="A003">
   <produit code="OU152"> Lessive </produit>
   <produit code="AH070"> Essuie-tout </produit>
 </service>
</magasin>
```


Déclaration d'une entité

- **Remplacement** d'une valeur (une chaîne de caractères) par un **symbole**, puis l'utilisation du symbole à la place de cette valeur.
- La syntaxe d'écriture d'une entité est la suivante : **<!ENTITY nom "VALEUR">**
- Il existe deux catégories d'entité: entité générales, entités paramètres.

Entité générale

- Les entités générales (sauf les entités prédéfinies) sont **définies dans la DTD** et **utilisées dans les documents XML correspondants**.

- Une entité générale est toujours invoquée sous la forme **&symbole;** au sein d'un document là où devrait apparaître le texte de remplacement associé.

Caractère	Entité
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

Les entités de caractères:

- Il existe des entités prédéfinies pour certains caractères spéciaux:
- Il existe des entités numériques du type: ***&#n;*** (où *n* est une valeur décimale)
  &#xh; (où h est une valeur hexadécimale)
- Exemple: π pour le symbole π.

Entité générale

Les entités internes:

- Elles sont utilisées pour ne pas avoir à taper de longues chaînes de caractères plusieurs fois dans un document.
- Elle sont déclarées dans les DTD internes.

- Exemple: **<!ENTITY UnivIK "Université de relizane">** **--DTD**

<titre> &UnivIK; vous souhaite bon courage dans les examens **</titre>** **--XML**

Entité générale

Les entités externes:

- Dans ce cas le texte de remplacement est défini dans un autre fichier.

- **<!ENTITY nom_entité SYSTEM "uri_remplacement">**

- Exemple:
```
<!ENTITY chap1 SYSTEM "chapitre1.xml">
<!ENTITY chap2 SYSTEM "chapitre2.xml">
  <livre>
    &chap1; <!--Inclusion du fichier chapiter1.xml -->
    &chap2; <!--Inclusion du fichier chapiter2.xml -->
  </livre>
```


Entité paramétré

- Les **entités paramètres** sont définies dans la DTD externe et utilisées dans la DTD elle-même.
- Elles permettent d'éviter des répétitions dans les définitions des éléments ou des attributs.
- Syntaxe: **<!ENTITY %nom_entité "Valeur" >**
- Exemple:

Définition de l'entité : **<!ENTITY %commun "niveau, couleur " >**
Utilisation de l'entité : **<!ELEMENT regtangle (%commun;, sommet+)>**
<!ELEMENT triangle (%commun;, sommet+)>

Il est équivalent au DTD suivant :
<!ELEMENT regtangle (niveau, couleur, sommet+)>
<!ELEMENT triangle (niveau, couleur, sommet+)>

Schéma XML ou XSD

- Un Schéma XML (XML Schema Definition, XSD) permet **la définition de schémas** (structure + type de données) des documents XML et facilite la communication entre applications.
- XML Schema est une **alternative** (successeur) pour les DTDs.
- **La différence entre les schémas XML et les DTDs:**

DTD	XML Schema
N'est pas une syntaxe XML	Basé sur la syntaxe XML
Difficile à étendre	Facilement extensible (héritage/importation)
Données textuelles non typées	Supporte les types de données
Ne permet pas de spécifier exactement le nombre d'occurrences d'un élément	Permet de spécifier exactement le nombre d'occurrences d'un élément
Ne supporte pas les espaces de noms	Supporte les espaces de noms

Structure XML schéma

- Un schéma xml est un fichier xml avec extension ' **.xsd** '.
- Comme tout document XML, un schéma xml commence par le prologue, puis l'élément racine **xsd:schema**.
- Syntaxe:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--Déclarations d'éléments, d'attributs et définitions de types -->
...
</xsd:schema >
```

- Avec : **xmlns:xsd = "http://www.w3.org/2001/XMLSchema"** : Indique que les éléments et types de données utilisés dans le schéma proviennent de l'espace de noms " ….2001/XMLSchema".
- Il spécifie également que les éléments et les types de données provenant de l'espace de noms "http://www.w3.org/2001/XMLSchema" doivent être préfixés par ' **xsd:** '.

Liaison de XSD dans un document XML

- Le référencement d'un schéma XML se fait au niveau de l'élément ***racine*** du fichier XML grâce à l'utilisation de **deux attributs**:

```
<?xml version="1.0" encoding="UTF-8"?>
<racine xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsd:SchemaLocation="chemin_vers_fichier.xsd">
….
</racine>

**********************************************************************
<?xml version="1.0" encoding="UTF-8"?>
<racine xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
xsd:noNamespaceSchemaLocation= " fichier.xsd">
….
</racine>
```


Déclaration d'un élément

- **Syntaxe:** **<xsd:element name="nom_element" type="type_element">**
- Exemples :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="date-naissance" type="xsd:date">
<xsd:element name="nombre-enfants" type="xsd:integer">
</xsd:schema >
```

- Les éléments peuvent être de type **simple** ou **complexe**.

Elément de type simple

- Un élément simple est un élément XML qui ne peut contenir que du **texte**. Il ne peut contenir aucun autre élément ou attribut.
- Le texte peut être de plusieurs types différents: Type prédéfinie, Type personnalisé.
- **Types prédéfinis : Numériques (**xsd:boolean, xsd:int, xsd:integer, xsd:decimal,…); **Chaines de caractères (**xsd:string, xsd:language (Code de langue comme fr),…); **Date & Heures (**xsd:time, xsd:date,…).
- **Type personnalisé** qu'on peut définir nous-même **: restrictions, union**, **list**, **extension.**

Elément de type simple

Création d'un nouveau type simple (restriction)

- La **restriction** (facette) permet d'obtenir un **type dérivé** à partir d'un **type de base** (héritage).
- La restriction d'un type est introduite par l'élément **xsd:restriction** dont l'attribut ***base*** donne le nom du type de base.
- On peut définir le nouveau type simple (**xsd:simpleType**) 'restriction' à l'aide de la syntaxe suivante :

```
<xsd:simpleType>
<xsd:restriction base="xsd:string">
……...
</xsd:restriction>
</xsd:simpleType>
```

- **Remarque** : on peut, aussi, faire **des restrictions** des types complexes. L'idée reste la même avec.

Elément de type simple

Création d'un nouveau type simple (restriction)

- **Restriction par longueur :** limiter la longueur d'une valeur dans un élément par ***maxLength*** et ***minLength (ou length)***.

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

- **Restriction par intervalle:** spécifier les limites supérieures / inférieures des valeurs numériques avec ***maxInclusive*(<=), *minInclusive*(>=), *maxExclusive*(<)** et ***minExclusive*(>)**.

```
<xsd:element name="annee">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="1970"/>
      <xsd:maxInclusive value="2050"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```


Elément de type simple

Création d'un nouveau type simple (restriction)

- **Restriction par énumération:** définit une liste de valeurs acceptables par ***enumeration***.

```
<xsd:element name="language" type="Langage"/>
  <xsd:simpleType>
    <xsd:restriction base="xsd:language">
      <xsd:enumeration value="ar"/>
      <xsd:enumeration value="an"/>
      <xsd:enumeration value="fr"/>
    </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

- **Restriction par motif:** définit la séquence exacte de caractères acceptable avec ***pattern***.

```
<xsd:element name="lettre">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([a-z][A-Z])+"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Attribut

- Tous les attributs sont déclarés comme des types simples. Si un élément a des attributs, il est considéré comme étant de type complexe.
- Syntaxe d'un attribut : **<xsd:attribute name=[nom attribut] type=[type attribut]/>**
- La déclaration des attributs doit être placée après la définition de type complexe.
- La présence d'un attribut peut être définie par l'attribut ***use***:
  - **optional** (employé par défaut).
  - **required** (obligatoire).
- Deux autres attributs, ***default*** et ***fixed***, servent à définir respectivement une valeur par défaut, si l'attribut est optionnel, et une valeur obligatoire.

```
<xsd:element name="personne">
 <xsd:complexType>
  <xsd:attribute name="nom1" type="xsd:string" use="required" fixed = "valeur" />
 </xsd:complexType>
</xsd:element>
```


Type complexe

- Un type complexe concerne le contenu d'un élément. Il caractérise une composition d'éléments ou d'attributs (d'où sa nature complexe).
- Dans un schéma, le type complexe nécessite toujours une balise **complexType**.

```
<xsd:element name="personne">
  <xsd:complexType>
   <!-- Déclaration des éléments ou bien attributs -->
  </xsd:complexType>
</xsd:element>
```

Il existe quatre types d'éléments complexes:
- Les éléments vides;
- Les éléments qui contiennent uniquement des éléments autres;
- Les éléments qui ne contiennent que du texte;
- Les éléments qui contiennent les deux autres éléments et le texte.

Type complexe: Elément vide

- **Eléments vides :** **<xsd:element name="personne" />**
- **Eléments vides avec attributs:**

```
<xsd:element name="personne">
  <xsd:complexType>
   <xsd:attribute name="nom" type="xsd:string" />
   <xsd:attribute name="prenom" type="xsd:string"/>
  </xsd:complexType>
 </xsd:element>
```


Type complexe: Elément ∋ Eléments

Eléments qui ne contiennent que d'autres éléments (utilisant d'indicateur d'ordre)**:**

- Une séquence d'éléments (***sequence***): l'élément *personne* contient un élément *nom* suivi d'un élément *prenom* et possède un attribut *id*.
- Une alternative d'éléments (***choice***): l'élément *personne* contient soit un élément *nom* ou un élément *prenom*.
- Une séquence d'éléments de présence obligatoire mais sans contrainte sur l'ordre (***all***).

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

```
<xsd:element name="personne">
  <xsd:complexType>
    <xsd:choice>
     <xsd:element name="nom" type="xsd:string"/>
     <xsd:element name="prenom" type="xsd:string"/>
    </xsd:choice> </xsd:complexType> </xsd:element>
```

Type complexe: Elément ∋ Textes

- **Eléments qui contiennent des textes et des attributs:**

```
<xsd:element name="personne">
  <xsd:complexType>
   <xsd:simpleType>
     <xsd:extension base ="xsd:string">
      <xsd:attribute name="nationalité" type="xsd:string" />
     </xsd:extension>
    <xsd:simpleType>
  </xsd:complexType>
  </xsd:element>
```

- Dans cet exemple, l'élément *personne* contient du *texte (type xsd:string)* et un attribut ayant pour nom *nationalité*.

Type complexe: Elément ∋ contenu mixte

- **Eléments qui contiennent des textes et d'autres éléments (contenus mixte):**

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

- **En XML:** **<personne>**
  Je suis <nom> Drif </nom> , <prenom> khalid </prenom> de l'Algerie
  </personne >

Indicateurs d'occurrence

- Il est possible de préciser le nombre minimal ou maximal d'occurrences d'un élément ou d'un groupe par les attributs **minOccurs** et **maxOccurs.**
- L'attribut **minOccurs** prend un entier comme valeur.
- L'attribut **maxOccurs** prend un entier ou la chaîne ***unbounded*** comme valeur pour indiquer qu'il n'y a pas de nombre maximal.
- La valeur par défaut de ces deux attributs est la valeur **1**.

```
<xsd:element name="personne">
  <xsd:complexType mixed="true">
   <xsd:sequence>
    <xsd:element name="nom" type="xsd:string" />
    <xsd:element name="prenom" type="xsd:string"
     minOccurs="1" maxOccurs ="3" />
   </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```


Exemple d'un schéma XML

Un simple document XML:

```
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Un fichier DTD:

```
<!DOCTYPE note [
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

Un schéma XML:

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:element name="note">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="to" type="xsd:string"/>
<xsd:element name="from" type="xsd:string"/>
<xsd:element name="heading“
type="xsd:string"/>
<xsd:element name="body" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Merci pour votre Attention

Vos Questions !!