

# 7 XML et les Données Semi-structurées

- L'apparition de XML (*eXtensible Markup Language*) (plus “évolué” que HTML) a mené au nouveau concept de *données semi-structurées*.
- XML : standard W3C d'échange de données sur le Web. Permet un échange sur un format standard, indépendamment des formats de stockage de ces données.
- Grande flexibilité.
- Multitude de standards associés:
  - Formats de Schémas : DTD et XML-schéma
  - Langages de Requête : XPATH, XQUERY (extension de XPATH),...
  - XSLT : notation pour transformer un document XML d'un format à un autre.etc.
- Lien avec le cours *Documents Structurés* ?

Parmi les applications BD :

- Intégration de Données (“mediateur”).
- Bases de Données en Biologie.

## 7.1 HTML

HTML (*Hyper Text Markup Language*) : un standard d'écriture de documents pour le Web.

HTML est un langage à **balises** (“étiquettes”). Ces balises sont **fixes**, à fonctions prédéfinies.

Les balises de HTML permettent de :

- **Mettre en forme un texte**

Ex. `<B> </B>`, `<I> </I>`, `<CENTER> </CENTER>`,.....

- **Créer des liens** (balises “amarres”).

Ex :

```
<A HREF="http://www.univ-evry.fr/">Universit\ 'e d'Evry Val d'Essonne </A>
```

Le rôle des balises autres que les “amarres” est celui de *présenter visuellement* du *texte* en un certain format.

Par exemple :

- `<B> bla </B>` sert à écrire **bla** à la place de bla
- `<I> bla </I>` sert à écrire *bla* à la place de bla
- `<CENTER> bla </CENTER>` sert à écrire

bla

à la place de :

bla

- `<H1> bla </H1>`, `<H2> blabla </H2>` `<H3> blablabla </H3>` servent à introduire des titres (des “sections”), par ordre d’importance décroissant.

# Exemple de document HTML : un morceau de ma page personnelle, écrite en HTML :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
  <TITLE> Serenella Cerrito </TITLE>
  <META NAME="GENERATOR" CONTENT="Mozilla/3.01Gold (X11; I;
SunOS 5.5 sun4m) [Netscape]">
</HEAD>
<BODY TEXT="#004D0F" BGCOLOR="#EEFFFF">

<H1 ALIGN=CENTER>
<IMG SRC="klee.tunisian-gardens.jpg" HEIGHT=370 WIDTH=370>
</H1>
</BR>

<H1 ALIGN=CENTER> <FONT COLOR="#FF7F50">
Serenella Cerrito </H1></FONT>

<H2><IMG SRC="blue-bullet.gif" HEIGHT=20 WIDTH=14> Current Position
<IMG SRC="home03.gif"></H2>

<H3>
Professor :</H3>
<UL>
<LI><A HREF="http://www.univ-evry.fr">Universit&eacute; d'Evry Val d'Essonne </A>
<LI> <A HREF = "http://www.ibisc.fr"> Member of I.B.I.S.C (previously : L.a.M.I.)</A>
<LI> <A HREF = http://www.ibisc.univ-evry.fr/Equipes/RMF> RMF Team </A>
</UL>
```

<H2><IMG SRC="blue-bullet.gif" HEIGHT=20 WIDTH=14 >Research  
<IMG SRC="idea.gif"></H2>  
<UL>  
<LI><H3><A HREF="nuovocv.ps">Curriculum Vitae, postscript file (may 2001 version). </A></H3><BR>  
</UL>  
<UL>  
<LI>  
<H3> Research Interests </H3>  
<UL>  
<LI><I>Logic and Automated Deduction</I></LI>  
<LI><I>Logic and Databases </I></LI>  
<LI><I>Logic and Logic Programming </I></LI>  
<LI><I>Logic and Functional Programming.</I></LI>  
</LI>  
</UL>

## 7.2 Limites de HTML

HTML n'est pas adapté à l'interrogation des données. Il permet de mettre en **FORME** un texte; il ne permet pas de **STRUCTURER** “logiquement” un contenu.

**Exemple 1** *Une organisation publie des données financières stockées dans une BD relationnelle et des pages web sont créées après une requête SQL. Une autre organisation veut une analyse financière de ces données, mais elle a accès seulement aux pages HTML. Pour cela, elle ne peut qu'écrire du logiciel qui transforme du texte HTML en une structure de données adaptée à l'analyse.*

- *Une petite modification du format d'un élément d'une page web peut casser ce logiciel !*
- *Même si on a besoin seulement de la valeur moyenne d'une colonne d'une table, on peut avoir besoin de charger une base entière via plusieurs requêtes de pages HTML.*



## 8 XML

XML : nouveau standard adopté par le World Wide Web Consortium (W3C) comme complément de HTML permettant un échange aisé de données de sur le web.

- Le but principal de XML n'est pas de décrire un format de texte, mais de **structurer** logiquement un contenu.
- Les balises ont le rôle de **classer des données** selon une hiérarchie **définie par l'auteur** du document XML.

- Avec XML, la mise en forme textuelle est effectuée dans une *feuille de style*, un document séparé qui associe des formes de présentations (texte en gras, en italique, centré, etc.) aux balises. Des feuilles différentes permettent des formattages différents du même document.
- Des outils permettent de convertir un document XML en HTML, afin de pouvoir afficher une page web.

## Example 2 Un petit document XML

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

```
<communication prior=' 'important' ' '>
```

```
<pour> Virginie </pour>
```

```
<sujet> Rappel </sujet>
```

```
<message> N'oublie pas de lire l'article
```

```
<lire> Lutz et al. 2002. </lire>
```

```
Il faut bien comprendre
```

```
<reflechir> la preuve de terminaison. </reflechir>
```

```
Rendez-vous <date> mercredi </date> <lieu> dans mon bureau </lieu>
```

```
</message>
```

```
<signature> Serena </signature>
```

```
</communication>
```

## Suite de l'exemple

Résultat de la mise en forme grce à une feuille de style :

Priorité : important

Pour : Virginie

Sujet : Rappel

N'oublie pas de lire l'article *Lutz et al. 2002*. Il faut bien comprendre **la preuve de terminaison**. Rendez-vous **mercredi** *dans mon bureau*

Serena

## Suite de l'exemple

Résultat de la mise en forme avec une **autre** feuille de style :

Priorité : IMPORTANT

Pour : VIRGINIE

Sujet : RAPPEL

N'oublie pas de lire l'article *Lutz et al. 2002*.

Il faut bien comprendre *la preuve de terminaison*.

Rendez-vous **mercredi dans mon bureau**

Serena

XML modélise des informations :

- En organisant les données en un *graphe d'objets complexes*
- En les structurant de façon plus *flexible* par rapport au au modèle relationnel ou objet : les données sont dites *semistructurées*

*graphe, objet complexe, flexible, semistructuré = ????*

⇒ Voir la suite...

## 8.1 Syntaxe de base de XML

La composante essentielle est l'*élément*, un morceau de document delimité par une balise d'ouverture (ex. `<toto>`) et une de fermeture (ex. `</toto>`).

Un élément peut contenir du texte, des autres éléments (→ “objet complexe”), ou un mélange des deux.

- Les balises (leur noms) **sont définies par les utilisateurs.**
- Elles n'ont pas de signification prédéfinie : elles indiquent seulement **comment structurer le document sous forme de arbre** (ou, plus généralement, de graphe) !!

### Example 3

```
<personne>  
<nom> Alan </nom>  
<age> 42 </age>  
<email> agb@abc.com </email>  
</personne>
```

*Ici on a :*

- *Un élément complexe de “sorte” (“type”) personne, qui consiste d’un triplet d’éléments ayant les “sortes” nom,age,email.*
- *Un élément Alan de “sorte” nom*
- *Un élément 42 de “sorte” age*
- *Un élément agb@abc.com de “sorte” email*



## Suite de l'exemple

Le contenu de ce document peut être représenté :

- Soit par un arbre où les *noeuds internes* sont étiquetés par les balises.
- Soit par un arbre où les *arcs* sont étiquetés par les balises.

FIGURES AU TABLEAU

## Example 4

<gens>

<personne>

<nom> Alan </nom>

<age> 42 </age>

<email> agb@abc.com </email>

</personne>

<personne>

<nom> Patricia </nom>

<age> 36 </age>

<email> ptn@abc.com </email>

</personne>

</gens>

Remarque : on peut utiliser plusieurs éléments ayant la même balise pour représenter une collection.

Dans l'exemple 4, une entité de “sorte” **gens** est une collection de personnes...

A nouveau, on peut représenter ces informations sous forme d'un arbre, avec 2 possibilités.

## Example 5

```
<biblio>
<livre>
<auteurs>
<nom> Abiteboul </nom>
<nom> Bunemann </nom>
<nom> Suciu </nom>
</auteurs>
<titre> Data on the Web </titre>
<edition>
<nom-ed> Morgan Kaufman </nom-ed>
<adresse-edition>
<rue-ed> 340, Pine Street </rue-ed>
<ville-ed> San Francisco </ville-ed>
<etat-ed> California </etat-ed>
<pays-ed> USA </pays-ed>
</adresse-edition>
</edition>
</livre>
<livre>
.
.
.
</livre>
</biblio>
```

## 8.2 Pourquoi “objet complexe” ?

Comparer avec les BD relationnelles (en première forme normale), où le domaine de tout attribut contient seulement des valeurs atomiques, et toute “entité est plate” :

nom	titre	nom-ed	rue-ed	ville-ed	etat-ed	pays-ed
Abit	Data on the Web	Kauf	340, Pine St.	S. Fr.	Cal	USA
Bune	Data on the Web	Kauf	340, Pine St.	S. Fr.	Cal	USA
Suciu	Data on the Web	Kauf	340, Pine St.	S. Fr.	Cal	USA
⋮						

Dans le document XML de l'exemple 5, un livre est un objet **complexe**, composé d'une séquence d'auteurs, d'un titre et d'une adresse (comme dans le BD à objet). La première et la troisième composantes sont elles mêmes des objets complexes.

## 8.3 Pourquoi “semistructuré” ?

Un premier élément de réponse :

un objet complexe peut avoir des composantes optionnelles, le “schéma” de la base n’est pas rigide.

Différence par rapport au modèle relationnel, où le nombre d’attributs du schéma d’une relation est fixé en avance.

## Example 6 *Un livre peut éventuellement être stocké avec son prix; le champ pays-ed est aussi optionnel :*

```
<biblio>
<livre>
<auteurs>
<nom> Abiteboul </nom>
<nom> Bunemann </nom>
<nom> Suciu </nom>
</auteurs>
<titre> Data on the Web </titre>
<edition>
<nom-ed> Morgan Kaufman </nom-ed>
<adresse-edition>
<rue-ed> 340, Pine Street </rue-ed>
<ville-ed> San Francisco </ville-ed>
<etat-ed> California </etat-ed>
<pays-ed> USA </pays-ed>
</adresse-edition>
</edition>
<prix-en-euros> 44 <prix-en-euros>
</livre>
<livre>
<auteurs>
<nom> Gardarin </nom>
</auteurs>
<titre> Internet/Intranet et Bases de Donn\`ees </titre>
<edition>
<nom-ed> Eyrolles </nom-ed>
<adresse-edition>
<rue-ed> 61, Bld Saint Germain </rue-ed>
<ville-ed> Paris </ville-ed>
```

<etat-ed> France </etat-ed>  
</adresse-edition>  
</edition>  
</livre>  
</biblio>



Dans les exemples vus jusqu'à ici, les données sont organisées en arbres. Les références produisent des **graphes**.

On peut faire référence à un sommet déjà existant dans le graphe car on peut associer à un *identificateur* à chaque élément.

Pour “pointer” vers un élément ayant identificateur, disons **cle** (nom choisi par l'auteur), on exploite l'existence des *attributs* XML.

En général, un attribut XML sert à définir une **propriété des données**; sa valeur est une chaîne de caractères. (Dans l'exemple 2, **prior** était un attribut).

La syntaxe générale de la déclaration d'attributs est :

```
<balise attribut1=valeur1 ... attributN = valeurN> ... </balise>
```

Par ex. :

```
<nom langue=français> Abiteboul </nom>
```

```
<nom langue=anglais> Bunemann </nom>
```

Pour identifier un élément il suffit d'utiliser un attribut dont le type déclaré (où ?) est ID :

`<balise attribut=valeur> </balise>` marche, à condition que `attribut` soit de type ID.

La syntaxe abrégée `<balise attribut=valeur/>` est aussi possible.

Par ex. :

```
<Livre ISBN="isbn-95456255"/>
```

C'est dans le schéma du document XML (voir après) que l'on on déclare l'attribut ISBN comme ayant le type ID.

Pour faire **référence** à un élément on utilise un attribut de type IDREF :

```
<balise attributRef= identificateur> </balise>
```

La syntaxe abrégée

```
<balise attributRef= identificateur /balise>
```

est aussi possible.

Par ex :

```
<Edition Editeur="LinuxFrench Edition1" EditeurRef= "LFNET" />
```

Ici, on déclarera l'attribut EditeurRef comme ayant le type IDREF.

Un élément de la forme :

```
<balise attributRef= identificateur /balise>
```

où attributRef est de type IDREF n'a pas de contenu. Il est dit dit *élément vide*.

## Example 7

```
<geographie-USA>
</etats>
<etat>
<etat cle = ''e1''>
<code-etat> IDA </code-etat>
<nom-etat> Idaho </nom-etat>
<capitale ref-cap = ''v1' />
<villes-dans ref-a-villes = ''v1'' />
<villes-dans ref-a-villes = ''v3'' />
...
</etat>
<etat>
...
</etat>
</etats>
<villes>
<ville>
<ville iden = ''v1''>
<code-ville> BOI </code-ville>
<nom-ville> Boise </nom-ville>
<etat-de-la-ville ref-a-etat = ''e1'' />
</ville>
<ville>
<ville iden = ''v2''>
<code-ville> CCN </code-ville>
<nom-ville> Carson City </nom-ville>
<etat-de-la-ville ref-a-etat = ''e2'' />
</ville>
<ville>
<ville iden = ''v3''>
<code-ville> MO </code-ville>
```

```
<nom-ville> Moscow </nom-ville>  
<etat-de-la-ville ref = ''e1'' />  
</ville>  
...  
</villes>  
</geographie-USA>
```

La possibilité de faire des références fait passer de la structure de *arbre* à celle plus générale de *graphe* orienté avec une racine.

Figure au tableau.

XML permet de mélanger des données textuelles et des sous-éléments au sein d'un élément :

**Exemple 8** <personne>

Voici mon meilleur ami

<nom> Alan </nom>

<age> 42 </age>

Je ne suis pas sûre de l'adresse e-mail suivante~:

<email> agb@abc.com </email>

</personne>

*Pas naturel du point de vue BD, mais du à l'origine de XML comme langage de documents hyper-texte.*



## 8.4 Schémas pour des documents XML

Deux formats :

1. Un *DTD* (**D**ocument **T**ype **D**efinition)
2. Un *XML-schema*, qui a une structure de typage plus riche.

### 8.4.1 Les DTD

Un DTD peut être vu comme une sorte de schéma pour les données XML. Il est **optionnel**  $\Rightarrow$  données **semistructurées**.

Une document XML qui, en outre d'être syntaxiquement correct, a un DTD, et le respecte, et dit *valide*.

## Syntaxe des DTD

Un document XML est composé de deux lignes initiales optionnelles suivie par une suite des éléments. La première des deux lignes initiales indique la version de XML utilisée, la deuxième contient le DTD.

```
<? xml version='1.0'?>  
<?DOCTYPE nom [Declarations-de-Type]>  
<nom> ... </nom>
```

La balise `nom` est la balise racine. `Declarations-de-Type` est une suite  $Déclaration_1, \dots, Déclaration_n$

où toute déclaration introduit le nom d'un élément et sa "sorte", c.à.d une description de la "forme de son contenu".

## Syntaxe d'une Déclaration du DTD

Pour le moment, ignorons les déclarations des types des attributs.

Chaque déclaration du DTD est constituée du symbole <, puis de la chaîne de caractères !ELEMENT, puis d'une balise, puis d'un modèle de contenu, et, enfin, le délimiteur de fin > :

```
<!ELEMENT balise modèle_contenu>
```

Il y a cinq sortes différents de modèles de contenu.

## Modèles de contenu dans une déclaration d'un DTD

1. Contenu vide : `<!ELEMENT balise EMPTY >`
2. Pas de contraintes sur le contenu : `<!ELEMENT balise ANY>`.  
(NB : données “semistructurées” !)
3. Élément ne contenant que des données textuelles :  
`<!ELEMENT balise #PCDATA>`
4. Élément ne contenant que d'autres éléments : `<!ELEMENT balise motif >`  
où `motif` est une *expression XML-régulière* sur l'alphabet des noms des balises.
5. Éléments de contenu “mixte” : mélange à la fois d'éléments et de données textuelles.

## Les opérateurs des expression XML-régulières

- Le symbole `,` indique la concaténation.

*Exemple* : `chat,chien` signifie qu'un chien doit suivre un chat. L'ordre compte dans les documents XML (arbres ordonnés). (**pourquoi ??**)

- Le symbole `|` est le XOR logique.

*Exemple* : `chat | tortue | chien` signifie que soit un chat soit une tortue soit un chien est acceptable (mais un seul de ces animaux).

- Le symbole `?` rend l'expression immédiatement précédente optionnelle.

*Exemple* : `(chat,chien) ?` signifie que une suite d'un chat puis d'un chien peut être placée à cet endroit, ou omise.

## Les opérateurs des expression XML-régulières, suite

- Le symbole  $+$  signifie qu'une suite non vide d'éléments conformes à l'expression immédiatement précédente est requise.

*Exemple :*  $(\text{chat} \mid \text{chien})^+$  signifie qu'il doit y avoir un nombre non nul de chats et de chiens.

- Le symbole  $*$  signifie qu'une suite éventuellement vide d'éléments conformes à l'expression immédiatement précédente est requise.

*Exemple :*  $(\text{chat}, \text{chien})^*$  signifie que, à cet endroit, ou bien il n'y a rien du tout, ou alors il y a une suite de chats et chiens telle que tout chat est immédiatement suivi par un chien et tout chien est immédiatement précédé par un chat.

## Example 9

<!ELEMENT article

(titre, sous-titre?, auteur\*, (paragraphe|table|figures)+, bibliographie?)>

*Cette déclaration décrit le contenu d'un article comme étant composé d'un titre suivi éventuellement d'un sous-titre, puis de 0 ou plusieurs auteurs, puis d'une combinaison (non-vide) de paragraphes, tables et figures, puis, éventuellement, d'une bibliographie.*



**Exemple 10** *Un exemple simple de DDT.*

```
<!DOCTYPE gens [  
<!ELEMENT gens (personne)*>  
<!ELEMENT personne (nom, age, e-mail)>  
<!ELEMENT nom (#PCDATA)>  
<!ELEMENT age (#PCDATA)>  
<!ELEMENT e-mail (#PCDATA)>  
>
```

*Le document de l'exemple 4 est valide par rapport à ce DTD.*

En résumant, il y a plusieurs raisons pour lesquelles on peut qualifier des données représentées dans un document XML comme étant *semi-structurées* :

1. Le document n'a pas de schéma (DTD ou XML-schéma). Dans ce cas, on a juste du texte, que l'on ne sait pas comme interroger ! (Sauf par recherche de mot clé, comme pour les documents HTML)
2. Le document a un schéma. Mais :
  - (a) Une déclaration de la forme `<!ELEMENT balise ANY>` ne donne aucune information sur la structure.
  - (b) Une déclaration comportant `?` prévoit l'optionalité d'une balise B dans le motif associé à une balise A. (Mais : penser aux valeurs nulles dans les SGBD relationnels...)

A la place d'inclure le DTD dans le document, on peut aussi le sauver dans un fichier séparé, qui peut être placé à une URL différente. Ceci permet à différents sites web de partager un unique schéma.

## Déclaration d'attributs dans un DTD

Un DTD permet aussi de déclarer le type des attributs. Par exemple ID est le type des attributs permettant de donner des identificateurs aux éléments (voir l'exemple 7 : `cle` est de type ID). Si un attribut  $A$  est déclaré comme ayant le type IDREF, ceci indique que la valeur de  $A$  est un identificateur d'un élément (l'élément pointé”).

Le mot-clé ATTLIST est utilisé pour déclarer le type d'une liste d'attributs.

En outre de déclarer le type des attributs, on décrit leur comportement; le mots-clés #REQUIRED, #IMPLIED indiquent, respectivement, si un attribut est obligatoire ou optionnel.

Syntaxe d'une déclaration de types pour une liste d'attributs :

```
<!ATTLIST nomatt1 typeatt1 descratt1, ..., nomattN typeattN descrattN>
```

## Exemple 11 *Un DDT pour les données de l'exemple 7 :*

```
<!DOCTYPE geographie-USA [  
<!ELEMENT geographie-USA (etats|villes)*>  
<!ELEMENT etats (etat)*>  
<!ELEMENT etat (code-etat,nom-etat,capitale,villes-dans*)>  
<!ATTLIST etat cle ID #REQUIRED>  
<!ELEMENT code-etat (#PCDATA)>  
<!ELEMENT nom-etat (#PCDATA)>  
<!ELEMENT capitale EMPTY>  
<!ATTLIST capitale ref-cap IDREF #REQUIRED>  
<!ELEMENT villes-dans EMPTY>  
<!ATTLIST villes-dans ref-a-villes IDREFS #REQUIRED>  
<!ELEMENT villes (ville)*>  
<!ELEMENT ville (code-ville,nom-ville,etat-de-la-ville)>  
<!ATTLIST ville iden ID #REQUIRED>  
<!ELEMENT code-ville (#PCDATA)>  
<!ELEMENT nom-ville (#PCDATA)>  
<!ELEMENT etat-de-la-ville EMPTY>  
<!ATTLIST etat-de-la-ville ref-a-etat IDREF #REQUIRED>  

```

*Un défaut des DTD : on ne peut pas déclarer que les sortes des valeurs de l'attribut `ref-cap`, par exemple, sont des villes. Les DTD n'offrent pas un typage adéquat des données semiestructurées.*

## 8.5 XML Schema

### Preliminaires

Un *espace de noms* XML est un standard W3C pour attribuer de façon unique des noms à certains éléments ou attributs d'un document XML.

Exemple d'ambiguïté : un noeud client et un noeud produit ont le même fils “num-ID”  $\Rightarrow$  problème pour les références.

Tout type atomique pre-défini XML appartient à un espace de noms :

- l'espace de noms de XML-schema : <http://www.w3.org/2001/XMLSchema> (le nom est prefixé par `xs`).
- l'espace de noms de XQuery : <http://www.w3.org/2003/11/xpath-datatypes> (le nom est prefixé par `xdt`).

## 8.6 Construction d'un schema XML

*Le contenu de cette section est fortement inspiré par le cours de Gilles Cagnon, disponible sur le web :*

`http://gilles.chagnon.free.fr/cours/xml/schema.html#introtdt`

mais je suis la seule responsable des erreurs éventuelles de cette section.

Un fichier Schema XML est lui-même un document XML.

Il commence par un prologue, et a un élément racine.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<xsd:schema xmlns:xsd='http://ww3.w3.org/2000:10/XMLSchema'>
<!-- declarations d'elements, d'attributs et de types ici -->
</xsd:schema>
```

Racine: l'élément `xsd:schema`.

L'attribut `xmlns:xs` fait référence à un espace de noms. Tout les éléments commençant par `xsd` sont référencés à la URL

`http://ww3.w3.org/2000:10/XMLSchema`

qui fixe la signification de certaines expressions.



## Déclaration d'éléments

Utilisation de la balise `xsd:element`. Par ex :

```
<?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>
<!-- declarations de types ici -->
</xsd:schema>
```

Declaration de 2 elements, `contacts` et `remarque`. Chaque élément est typé : `contacts` a un type *complexe* (il pourra contenir des éléments enfants) tandis que `remarque` a un type simple pre-défini (`xsd:string`).

## Déclaration d'attributs

Utilisation de la balise `xsd:attribut`.

Les déclarations d'attributs doivent être placées après les définitions des types complexes (par contre, dans un DTD : ordre des déclarations sans importance).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<xsd:schema xmlns:xsd='http://ww3.w3.org/2000:10/XMLSchema'>
<xsd:element name='contacts' type='typeContacts' />
<xsd:element name='remarque' type='xsd:string' />
<!-- declarations de types ici -->
<xsd:complexType name = 'typeContacts'>
<!-- declarations du mod\ele de contenu ici -->
<xsd:attribute name='maj' type='xsd:date' />
</xsd:complexType>
</xsd:schema>
```

NB : Attribut `maj` : date de la dernière mise à jour de la liste des contacts. `xsd:date` : type simple XML.

Un attribut peut être déclaré :  
obligatoire : `required`,  
optionnel : `optional`,  
ayant une valeur par défaut.

Par ex.

```
<xsd:attribute name='maj' type='xsd:date' use='optional'  
default= 2006-03-11 />
```

Quand on déclare un type complexe, ayant plusieurs composantes, d'un élément (ci-dessous : `livre`) il faut commencer par déclarer les éléments et les attributs de type simple, à cause des possibilité des "références".

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

**NB** : "*référence*", ICI : un livre est un élément de type complexe, c.à.d. une séquence de pages et d'un auteur.

## Types Simples de XML Schema

Il en a beaucoup. Par ex., string, boolean, positiveinteger, cdata, date, ID, IDREF, idrefs, NMTOKEN (une lettre, une chiffre, un point, un tiré, une virgule..)

Les types **listes** sont des suites de types simples.

XML Schema possède 3 types de listes intégrées :

1. NMTOKENS : suites finites et non-vides de NMTOKEN séparés par un espace.
2. ENTITY : notion XML un peu plus complexe que ELEMENT. Passons.
3. IDREFS (déjà vu avec les DTD)

## Types Complexes

Les sequences :

```
<xsd:complexType>  
  <xsd:sequence>  
    <xsd:element name =''nom'' type =xsd:string'' />  
    <xsd:element name =''prenom'' type =xsd:string'' />  
    <xsd:element name =''dateNaissance'' type =xsd:date'' />  
    <xsd:element name =''adresse'' type =xsd:string'' />  
    <xsd:element name =''e-mail'' type =xsd:string'' />  
  </xsd:sequence>  
</xsd:complexType>
```

NB : date, string : types atomiques de XML Schema

## Types Complexes, Suite

Modifier la déclaration de type précédente en stipulant que l'on doit indiquer soit l'adresse d'une personne soit son adresse mail : utiliser un élément `xsd:choice`

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name =''nom'' type =xsd:string'' />
    <xsd:element name =''prenom'' type =xsd:string'' />
    <xsd:element name =''dateNaissance'' type =xsd:date'' />
    <xsd:choice >
      <xsd:element name =''adresse'' type =xsd:string'' />
      <xsd:element name =''e-mail'' type =xsd:string'' />
    </xsd:choice >
  </xsd:sequence>
</xsd:complexType>
```

## Types Complexes, Suite

Vraie nouveauté par rapport aux DTD. On peut indiquer que les éléments enfants doivent apparaître une fois (ou pas du tout), et dans n'importe quel ordre. L'élément `xsd:all` doit être un enfant direct de `xds:complexType`

```
<xsd:complexType>
  <xsd:all>
    <xsd:element name =''nom'' type =xsd:string'' />
    <xsd:element name =''prenom'' type =xsd:string'' />
    <xsd:element name =''dateNaissance'' type =xsd:date'' />
    <xsd:element name =''adresse'' type =xsd:string'' />
    <xsd:element name =''e-mail'' type =xsd:string'' />
  </xsd:all>
</xsd:complexType>
```

Chacun de ses éléments peut apparaître une fois ou pas de tout (comparer à ? dans le DTD), mais l'ordre des éléments n'a pas d'importance ( $\neq$  DTD !).