

# Module MLBDA Master Informatique Spécialité DAC

---

COURS 5 – XSCHEMA

# Objectifs

---

## Définir un nouveau langage de schémas XML :

- Reprendre les acquis des DTD
- Réutiliser des définitions de structures pour différents éléments (type)
- Exprimer des contraintes plus fortes
- Utiliser XML pour exprimer aussi les schémas

## Et aussi

- Prendre en compte les *espaces de noms*
- Extensibilité
- Notion de clef plus expressive et précise
- Dérivation de types (restriction, extension)

# DTD

---

## DOCUMENT XML

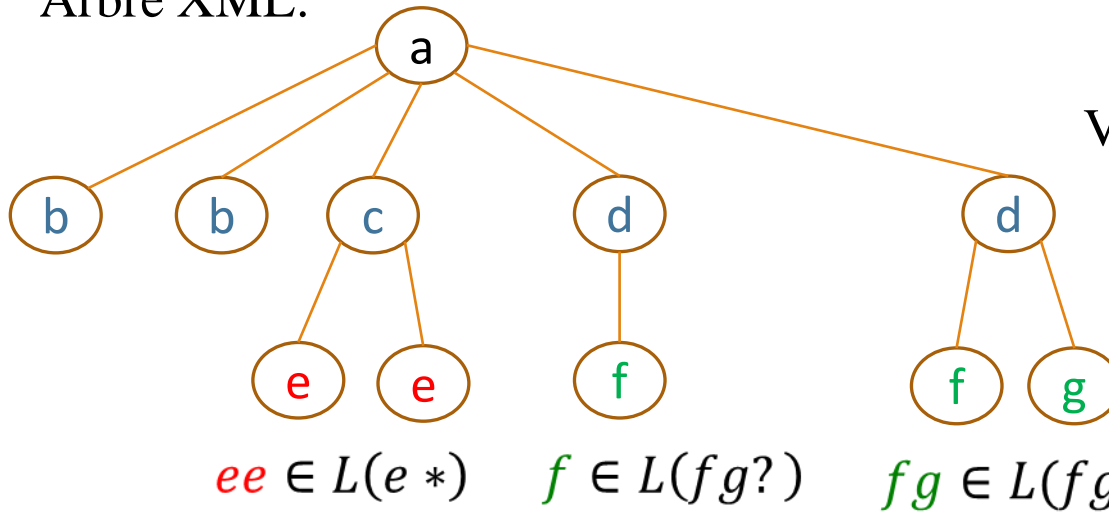
```
<?xml version="1.0" ?>
<note>
  <pour>Jean</pour>
  <de>Marie</de>
  <sujet>reunion</sujet>
  <texte>demain 15h</texte>
</note>
```

## DTD

```
<!ELEMENT note
(pour,de,sujet,texte)>
<!ELEMENT pour (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT sujet (#PCDATA)>
<!ELEMENT texte (#PCDATA)>
```

# DTD et grammaires

Arbre XML:



Validation :

$bbcdd \in L(b * cd *)$

DTD :

```
<!ELEMENT a (b*cd*) >  
<!ELEMENT b EMPTY >  
<!ELEMENT c (e*) >  
<!ELEMENT d (fg?) >
```

Expression régulières:

- $L(b*cd*) = \{c, bc, cd, bcd, bbc, cdd, bbbc, bbcd, \dots\}$
- $L(e^*) = \{\epsilon, e, ee, \dots\}$
- $L(fg?) = \{f, fg\}$

# DTD comme grammaire

---

Une DTD est une couple (E,R) où

- E est un ensemble de noms d'éléments
- S est un élément de E (éléments racine)
- $R = \{e \rightarrow r\}$  est un ensemble de règles telles que
  - $e \in E$  est un nom d'élément
  - r est une expression régulières sur les noms d'éléments

DTD

```
<!ELEMENT officiel (#PCDATA | cinema | film)*>  
<!ELEMENT cinéma (nom, adresse, (séances))*>
```

Grammaire

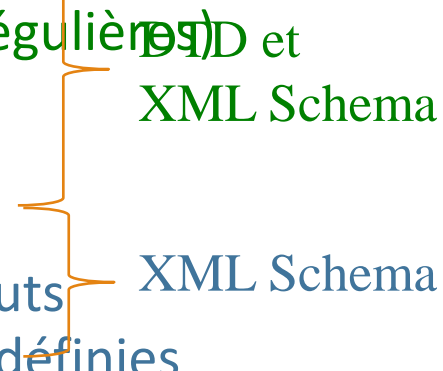
```
officiel → (#PCDATA|cinéma|film)*  
cinéma → nom adresse (séance)*
```

# XML Schema

---

Un schéma XML est un **document XML** avec un élément racine `xs:schema` (`xs` désigne l'espace de nom de XML Schema).

XML Schema permet de définir

- les éléments fils et leur ordre (expressions régulières)
  - les attributs
  - les valeurs par défaut et les valeurs fixes
  - les clefs et références typées
  - les **types de données** des éléments et attributs
  - les **espaces de noms** des balises utilisées et définies
- 
- XML Schema
- XML Schema

<http://www.w3.org/TR/xmlschema-2/>

# Schéma XML : Types et éléments

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
  targetNamespace="http://www.w3schools.com"
```

```
  xmlns="http://www.w3schools.com"
```

```
  elementFormDefault="qualified">
```

```
<xs:element name="note">
```

```
  <xs:complexType>
```

```
    <xs:sequence>
```

```
      <xs:element name="pour" type="xs:string"/>
```

```
      <xs:element name="de" type="xs:string"/>
```

```
      <xs:element name="sujet" type="xs:string"/>
```

```
      <xs:element name="texte" type="xs:string"/>
```

```
    </xs:sequence>
```

```
  </xs:complexType>
```

```
</xs:element>
```

```
</xs:schema>
```

```
<!ELEMENT note (pour,de,sujet,texte)>
```

```
<!ELEMENT pour (#PCDATA)>
```

```
<!ELEMENT de (#PCDATA)>
```

```
<!ELEMENT sujet (#PCDATA)>
```

```
<!ELEMENT texte (#PCDATA)>
```

# Déclaration d'Eléments (1)

---

Un élément a un *modèle de contenu simple ou complexe*

## Modèle de contenu simple

- *Types simples* : string, boolean, float, timeInstant, timePeriod, month, date, ...
- Exemple :

```
<xs:element name="pour" type="xs:string"/>
```

```
<xs:element name="datenais" type="xs:date"/>
```

On peut déclarer des valeurs par défaut ou des valeurs fixes :

```
<xs:element name="couleur" type="xs:string" default="bleu"/>
```

```
<xs:element name="couleur" type="xs:string" fixed="bleu"/>
```



# Déclarations d'Eléments (2)

---

## Modèle de contenu complexe : <complexType>

- Types complexes: <sequence>, <choice>, <all>
- Contenus mixtes (attribut mixed="true")

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="pour" type="xs:string"
        minOccurs="1"
        maxOccurs="unbounded" />
      <xs:element name="de" type="xs:string"/>
      <xs:element name="sujet" type="xs:string"/>
      <xs:element name="texte" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Types complexes: séquence, choice, all

---

<code>&lt;xs:sequence&gt;</code>	: séquence ordonnée d'éléments	: (A,B,C)
<code>&lt;xs:choice&gt;</code>	: choix parmi un ensemble d'éléments	: (A B C)
<code>&lt;xs:all&gt;</code>	: séquence non ordonnée	

```
<xs:element name="ident">
  <xs:complexType>
    <xs:choice>
      <xs:element name="raisonsociale" type="xs:string"/>
      <xs:element name="identité" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

# *minoccurs et maxoccurs*

---

	<b>minoccurs</b>	<b>maxoccurs</b>
<code>&lt;xs:sequence&gt;</code>	Entier $\geq 0$	Entier $\geq$ minoccurs
<code>&lt;xs:choice&gt;</code>	Entier $\geq 0$	Entier $\geq$ minoccurs
<code>&lt;xs:all&gt;</code>	0 ou 1	1

```
<xs:element name="ident">
  <xs:complexType>
    <xs:all>
      <xs:element name="prenom" type="xs:string"
        minoccurs = 0/>
      <xs:element name="nom" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Valeurs par défaut : 1

# Éléments (3)

---

Le **contenu** peut être défini de façon indépendante de l'élément, et référencé par l'attribut **ref** (réutilisation de déclarations d'éléments).

Les deux façons de définir les éléments (directement ou par référence) sont équivalentes.

```
<xs:element name="NOM" type="xs:string"/>
<xs:element name="IDENTITE" >
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="NOM"/>
      <xs:element name="PRENOM" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Déclaration d'Attributs

---

## Déclaration d'un attribut

```
<xs:attribute name="langue" type="xs:string">
```

Les attributs sont typés (toujours des **types simples**)

## Contraintes

- Optionalité : use="optional" | "required" | "prohibited"
- Valeur par défaut : default="string"
- Valeur constante : fixed="10"

Lorsqu'il y a une valeur fixe ou une valeur par défaut, **use** doit avoir la valeur optional (c'est la valeur par défaut).

# Déclaration d'Attributs

---

Les attributs font partie du type de l'élément:

```
<xs:element name="foo">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
    <xs:attribute name="bar" .../>
    <xs:attribute name="boo" .../>
  </xs:complexType>
</xs:element>
```

# Déclaration de Types

---

## Types simples prédéfinis :

- string, integer, positiveInteger, negativeInteger, boolean, time, date, ID, IDREF, IDREFS, etc.

## Possibilité de définir des **nouveaux types** (*dérivation*)

- constructeurs de types complexes: [sequence](#), [choice](#), [all](#)
- types simples dérivés : contraindre les valeurs ([restriction](#), [facettes](#))
- types complexes dérivées : contraindre la structure ([restriction](#), [extension](#))

Un type peut être identifié ou anonyme.

# Type complexe identifié

---

```
<xs:complexType name="caType">
  <xs:sequence>
    <xs:element name="adresse" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="datecreation" type="xs:date"/>
  <xs:attribute name="datemaj" type="xs:date"/>
</xs:complexType>

<xs:element name="carnetadresse" type="caType"/>
```



# Type simple identifié et dérivé par restriction

---

Nouveau type par spécialisation d'un type prédéfini (string, date, etc.)

```
<xs:simpleType name="cpType">  
  <xs:restriction base="xs:string">  
    <xs:pattern value="d\{(5)"/>  
  </xs:restriction>  
</xs:simpleType>  
<xs:element name="CodePostal" type="cpType"/>
```

# Type complexe identifié

---

```
<xs:complexType name="typeSociete">
  <xs:sequence>
    <xs:element name="siret" xs:type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:element name="societe" type="typeSociete"/>
```

# Remarque

---

Une déclaration d'élément peut avoir un *attribut type* (référence vers un type identifié) **ou** un *élément fils* **complexType** (type anonyme), mais pas les deux simultanément:

```
<xs:element name="A" type="foo">
```

```
<xs:complexType>
```

```
...
```

```
</xs:complexType>
```

```
</xs:element>
```



# Type complexe mixte

---

Contenu mixte (mélange de texte et d'éléments)

```
<xs:element name "meteo" >
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="loc" type="xs:string"/>
      <xs:element name="heure" type="nonNegativeInteger"/>
      <xs:element name="temps" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Exemple: <meteo>*situation météo pour la* <loc>région parisienne</lo> *ce mercredi à*  
<heure>13h</heure> : <temps>temps frais et couvert</temps>.</meteo>

**Attention :** L'ordre des éléments <loc>, <heure>, <temps> doit être respecté.

**DTD:** <!ELEMENT meteo (#PCDATA|loc|heure|temps)\*> (l'ordre est ignoré)

# Types dérivés

---

## Notions d'héritage et de spécialisation

### Dériver :

- Ajouter ou contraindre des caractéristiques d'un type
- Définition d'un nouveau type résultant

### Dérivation par ***restriction***

- Réduction du champ de portée d'un type
- *Ajouter des contraintes* (maxOccurs= '1' , facette)

### Dérivation par ***extension***

- Réutilisation d'un type complexe
- *Ajouter des informations* supplémentaires (attribut, élément)

# Contraintes de types simples

---

Les *facettes* permettent de définir **un nouveau type simple** à partir d'un **type simple existant** (appelé le type de base), en spécifiant des valeurs pour une ou plusieurs facettes:

Exemple: le type **string** a six *facettes optionnelles* :

- **length, minLength, maxLength**
- **pattern** : expression régulière définissant la suite de caractères acceptable
- **enumeration** : liste de valeurs acceptables
- **whitespace** : traitements des espaces blancs (tab, espace, retour chariot)

# Facettes xs:length

---

```
<xs:simpleType name="TelephoneNumber">  
  <xs:restriction base="xs:string">  
    <xs:length value="8"/>  
    <xs:pattern value="\d{3}-\d{4}"/>  
  </xs:restriction>  
</xs:simpleType>
```

Crée un nouveau type dérivé de **xs:string** appelé **TelephoneNumber**

Les éléments de ce type ont des valeurs de type string

La longueur de la chaîne de caractère doit être exactement 8

La chaîne doit respecter le format ddd-dddd (où d est un chiffre)

# Facettes xs:enumeration

---

```
<xs:simpleType name="forme">
  <xs:restriction base="xs:string">
    <xs:enumeration value="cercle"/>
    <xs:enumeration value="triangle"/>
    <xs:enumeration value="carre"/>
  </xs:restriction>
</xs:simpleType>
```

Crée un nouveau type **forme**.

Un élément de ce type doit avoir comme valeur cercle, triangle ou carré.



# Facette xs:length et xs:pattern

---

```
<xs:complexType name="typeSociete">
  <xs:sequence>
    <xs:element name="siret">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:length value="16">
            <xs:pattern value="\d(6)\s\d(3)\s\d(5)"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="societe" type="typeSociete"/>
```

# Facettes pour xs:integer

```
<xs:simpleType name= "Temperature">
```

```
  <xs:restriction base="xs:integer">
```

```
    <xs:minInclusive value="-60"/>
```

```
    <xs:maxInclusive value="80"/>
```

```
  </xs:restriction>
```

```
</xs:simpleType>
```

8 facettes :

- totalDigitspattern
- whitespace
- enumeration
- maxInclusive, minInclusive
- maxExclusive, minExclusive.

# Utilisation de plusieurs facettes

---

Les facettes **pattern** et **enumeration** sont liées par un « OU », toutes les autres sont liées par « ET » (elles doivent toutes être vérifiées)

La forme doit être un cercle, un triangle **OU** un carré.

```
<xs:simpleType name="forme">
  <xs:restriction base="xs:string">
    <xs:enumeration value="cercle"/>
    <xs:enumeration value="triangle"/>
    <xs:enumeration value="carre"/>
  </xs:restriction>
</xs:simpleType>
```

Le numéro de téléphone doit avoir 8 chiffres, **ET** doit respecter la forme suivante ddd-dddd.

```
<xs:simpleType name="TelephoneNumber">
  <xs:restriction base="xs:string">
    <xs:length value="8"/>
    <xs:pattern value="\d{3}-\d{4}"/>
  </xs:restriction>
</xs:simpleType>
```

# Facettes de facettes

---

## TEMPERATURES

```
<xs:simpleType
  name= "Temperature">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="-60"/>
    <xs:maxInclusive value="80"/>
  </xs:restriction>
</xs:simpleType>
```

## TEMPERATURES EN FRANCE

```
<xs:simpleType
  name= "TemperatureEnFrance">
  <xs:restriction base= "Temperature">
    <xs:minInclusive value="-30"/>
    <xs:maxInclusive value="45"/>
  </xs:restriction>
</xs:simpleType>
```

# Attributs : énumération

---

Un attribut peut être restreint à un ensemble de valeurs de type simple:

```
<xs:attribute name="typetelephone">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="standard"/>
      <xs:enumeration value="fax"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

# Restriction de types complexes

```
<xs:complexType name="Publication">
```

```
  <xs:sequence>
```

```
    <xs:element name="Titre" type="xs:string" maxOccurs="unbounded"/>
```

```
    <xs:element name="Auteur" type="xs:string" maxOccurs="unbounded"/>
```

```
  </xs:sequence>
```

```
</xs:complexType>
```

```
<xs:complexType name="PublicationUniqueAuteur">
```

```
  <xs:complexContent>
```

```
    <xs:restriction base="Publication">
```

```
      <xs:sequence>
```

```
        <xs:element name="Titre" type="xs:string" maxOccurs="unbounded"/>
```

```
        <xs:element name="Auteur" type="xs:string"/>
```

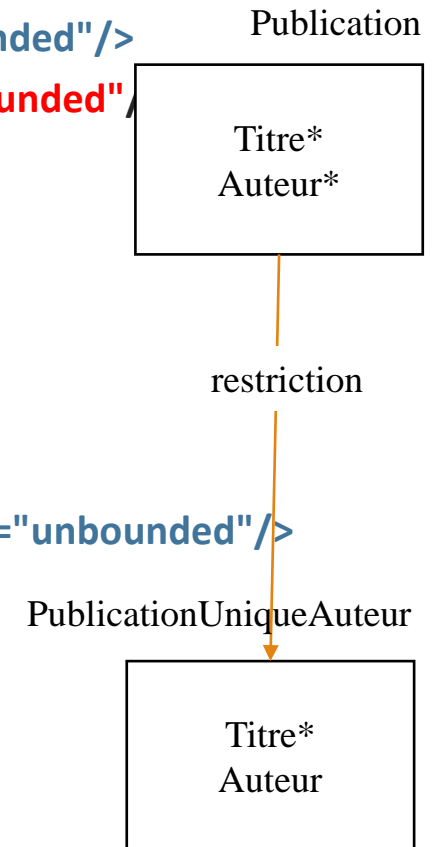
```
      </xs:sequence>
```

```
    </xs:restriction>
```

```
  </xs:complexContent>
```

```
</xs:complexType>
```

On répète les éléments du type de base qu'on veut garder (Titre).



# Extension de types complexes

```
<xs:complexType name="personneType">
```

- `<xs:sequence>`

- `<xs:element name="nom" type="xs:string"/>`

- `<xs:element name="datenais" type="xs:date"/>`

- `</xs:sequence>`

```
</xs:complexType>
```

```
<xs:complexType name="etudiantType">
```

- `<xs:complexContent>`

- `<xs:extension base="personneType">`

- `<xs:sequence>`

- `<xs:element name="cursus" type="xs:string"/>`

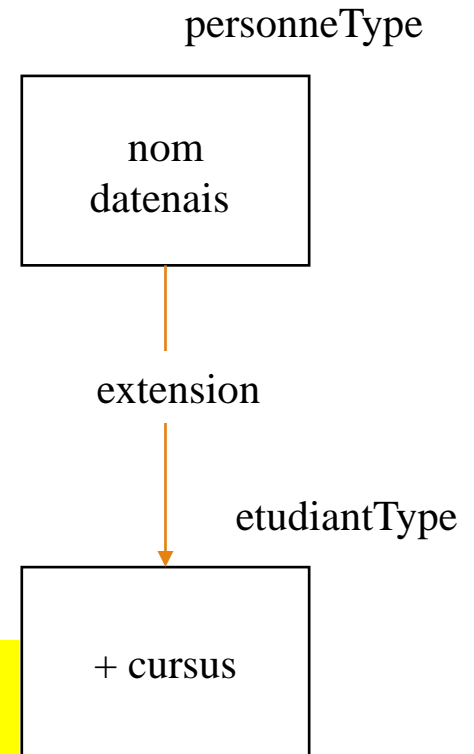
- `</xs:sequence>`

- `</xs:extension>`

- `</xs:complexContent>`

```
</xs:complexType>
```

On ne répète pas les éléments  
du type de base qu'on veut  
garder.



# Types de contenu de types complexes

---

## Types de contenu :

- simples : nœuds texte ou vide
- complexes : nœuds attributs et nœuds éléments

## Éléments à contenu vide (ne contenant que des attributs) :

```
<xs:element name="book">  
  <xs:complexType>  
    <xs:attribute name="isbn" type="isbnType"/>  
  </xs:complexType>  
</xs:element>
```

L'élément `<book>` est de **type complexe** avec un **contenu simple** (vide):

```
<book isbn="0123456789" />
```



# xs:simpleContent vs. xs:complexContent

- `<xs:simpleContent>` permet de définir des nouveaux **types complexes** par extension ou restriction d'un **type simple**.
- `<xs:complexContent>` permet de définir des nouveaux **types complexes** par extension ou restriction d'un **type complexe**.

```
<xs:complexType name="...">
  <xs:complexContent>
    <xs:extension base="X">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

X est un **type complexe**

```
<xs:complexType name="...">
  <xs:simpleContent>
    <xs:extension base="Y">
      ...
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Y est un **type simple**

# xs:simpleContent

---

Pour définir un élément ne contenant que du texte et des attributs, on *dérive* un **type complexe** avec un **type de contenu simple** (ici une extension du type string avec un attribut) en utilisant l'élément **xs:simpleContent** :

```
<xs:element name="book">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="isbn" type="isbnType" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

L'élément <book> peut aussi contenir du texte :

```
<book isbn="0123456789">Ce livre est épuisée </book>
```

# xs:complexContent

---

L'élément **complexContent** permet d'étendre ou de restreindre un **type complexe** :

```
<xs:complexType name="Publication">
  <xs:sequence>
    <xs:element name="Titre" type="xs:string" maxOccurs="unbounded"/>
    <xs:element name="Auteur" type="xs:string" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PublicationUniqueAuteur">
  <xs:complexContent>
    <xs:restriction base="Publication">
      <xs:sequence>
        <xs:element name="Titre" type="xs:string" maxOccurs="unbounded"/>
        <xs:element name="Auteur" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Catalogue" type="PublicationUniqueAuteur"/>
```

# Clés et références

---

## Dans les DTD:

- un élément ayant un attribut de type IDREF doit référencer un élément ayant un attribut de type ID.
- Le parser vérifie que la valeur de IDREF correspond à une valeur existante de ID.
- On ne peut pas définir des références « typés » : les attributs de type IDREF peuvent référencer tous les éléments avec un attribut ID

## Dans XML Schema:

- une clé est un élément `<xs:key>` qui définit les identifiants pour un ensemble d'éléments
- une références est un élément `<xs:keyref>` qui définit les éléments qui correspondent aux références vers une clé.
- On peut « typer » les références: les éléments A sont des références vers les éléments B

# Unicité et clés

---

Dans les DTD, une valeur d'attribut de type ID doit être unique dans tout le document, pour un élément donné.

XML Schema étend et généralise le mécanisme des ID/IDREF dans les DTD. Il permet de

- Garantir l'unicité du contenu d'un élément
- Garantir l'unicité d'attributs de type quelconque (pas forcément ID)
- Garantir l'unicité simultanée du contenu d'un élément et des attributs
- Distinguer les notions d'unicité et de clé

# Unicité et clés

---

Xschema permet de définir l'unicité avec `<xs:unique>` et `<xs:key>`.

Key: un élément ou un attribut (ou une combinaison des deux) défini avec key :

- doit toujours être présent (minOccurs doit être >0)
- ne peut pas avoir une valeur nulle (nillable="false")
- doit être unique

Unique : identique à key sauf que seule l'unicité est requise (pas l'existence).

Si les valeurs sont présentes, elles sont uniques.

Key implique unique, mais unique n'implique pas key

# Utilisation

---

L'élément `<xs:key>` doit être imbriqué dans un élément, et doit se trouver après le modèle de contenu et les déclarations d'attributs.

L'élément `<xs:selector>` est utilisé comme un fils de `<xs:key>` pour déterminer un ensemble d'éléments pour lesquels la clé s'applique.

L'élément `<xs:field>` est utilisé comme un fils de `<xs:key>` pour identifier l'élément ou l'attribut clé.

- Il peut y avoir plusieurs éléments `<xs:field>`.

# Exemple : definition de clés

---

```
<xs:element name="Librairie">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Livre" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Titre" type="xs:string"/>
            <xs:element name="Auteur" type="xs:string"/>
            <xs:element name="ISBN" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:key name="Lclef">
    <xs:selector xpath="Livre"/>
    <xs:field xpath="ISBN"/>
  </xs:key>
</xs:element>
```



# Exemple : definition de clés

```
<xs:element name="Librairie">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Livre" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Titre" type="xs:string"/>
            <xs:element name="Auteur" type="xs:string"/>
            <xs:element name="ISBN" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:key name="Lclef">
    <xs:selector xpath="Livre"/>
    <xs:field xpath="Titre"/>
    <xs:field xpath="Auteur"/>
  </xs:key>
</xs:element>
```

# Exemple : clés et références

```
<Librairie>
  <LesLivres>
    <Livre>
      <Titre>Le soleil des Scorta</Titre>
      <Auteur>Laurent Gaudé</Auteur>
      <Date>2004</Date>
      <ISBN>0-123-45678-9</ISBN>
    </Livre>
    ...
  </LesLivres>
  <Signatures>
    <Auteur>
      <Nom>Laurent Gaudé</Nom>
      <LivreAssigner>
        <ISBNRef>0-123-45678-9</ISBNRef>
      </LivreAssigner>
    </Auteur>
  </Signatures>
</Librairie>
```

XML Schema:

```
<xs:key name= "Lclef">
  <xs:selector xpath="LesLivres/Livre"/>
  <xs:field xpath="ISBN"/>
</xs:key>
<xs:keyref name="isbnRef" refer="Lclef">
  <xs:selector xpath="Signatures/Auteur/LivreAssigner"/>
  <xs:field xpath="ISBNRef"/>
</xs:keyref>
```

<xs:keyref> doit avoir autant d'éléments <field> que <xs:key>, ils doivent être dans le même ordre, et doivent avoir les mêmes types.

# Élément <unique>

---

L'élément <xs:unique> se définit de façon analogue à l'élément <xs:key>. Il a un élément fils <xs:selector> et un ou plusieurs éléments fils <xs:field>.

La seule différence est que **l'existence de l'élément désigné par <xs:field> n'est pas obligatoire.**

```
<xs:unique name="Lclef">  
  <xs:selector xpath="Livre"/>  
  <xs:field xpath="ISBN"/>  
</xs:unique>
```

Tous les éléments ISBN ont une valeur unique, mais un élément livre peut ne pas avoir de valeur pour ISBN.

# Espaces de noms

---

Importer, dans un document, des éléments ou des attributs contenus dans des entités externes, peut entraîner des conflits de noms.

Pour éviter ces conflits, on définit des *espaces de noms*

L'objectif est de pouvoir mélanger plusieurs vocabulaires au sein d'un même document.

Un espace de nom permet

- d'identifier la provenance de chaque élément et attribut,
- d'éviter les conflits de noms,
- de faire coopérer les noms des différentes structures XML : copier-coller des fragments de documents
- de réutiliser des déclarations

# Espace de noms (NameSpace)

---

Un espace de noms XML est une collection abstraite de noms d'éléments ou de noms d'attributs identifié par une URI

Exemples:

- XMLSchema: <http://www.w3.org/2001/XMLSchema>
- Dublin Core : <http://purl.org/dc/elements/1.1/>
- XSLT : <http://www.w3.org/999:XSL.Format>
- MathML : <http://www.w3.org/1998:Math/MathML>

Un élément ou attribut A d'un espace de nom N est désigné par un nom qualifié N:A compose de l'URI de l'espace de nom et du nom local

Exemples:

- <http://www.w3.org/2001/XMLSchema:element>
- <http://purl.org/dc/elements/1.1/author>

Au lieu d'utiliser les URI on peut associer des préfixes aux espace de nom :

- Un préfixes est déclaré comme attribut d'un élément et sa définition porte sur l'élément et son le sous-arbre.

# Déclaration de l'espace de nom

---

```
<xs:schema  
  xmlns:xs=http://www.w3.org/2001/XMLSchema  
  targetNamespace="http://www.w3schools.com"  
  xmlns="http://www.w3schools.com"  
  elementFormDefault="qualified">
```

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` indique l'espace de noms contenant les éléments et types de données utilisés dans le schéma (schema, element, sequence, complexType, string, etc.).
- `targetNamespace="http://www.w3schools.com"` indique l'espace de noms pour les éléments définis par ce schéma (note, de, pour, sujet, texte).

# Élément schema

---

`xmlns="http://www.w3schools.com"`

- indique l'espace de noms par défaut (le même que le `targetNamespace`). Utilisé lors de références (attribut `ref`) lorsqu'on ne précise pas l'espace de noms.

`elementFormDefault="qualified"`

- peut prendre les valeurs *qualified* ou *unqualified* (valeur par défaut).
- *qualified* : tous les éléments sont dans l'espace de noms cible.
- *unqualified* : seuls les éléments définis globalement (dans un élément directement fils de l'élément `xs:schema`) sont dans l'espace de noms cible. Les autres sont sans espace de noms.

# Référencer un schéma

---

Dans la racine du document XML à valider, on utilise l'attribut **schemaLocation**, qui indique l'espace de noms cible et l'adresse du schéma.

L'attribut **schemaLocation** se trouve dans l'espace de noms des instances de schémas

La valeur de schemaLocation est une suite d'URI séparés par des espaces.

Les URI vont par paires 'nsN schN' où nsN est l'espace de nom cible du schéma schN.

*schemaLocation='ns1 sch1 ns2 sch2 ... nsN schN'*



# Exemple

---

Fichier base.xml

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

```
<base xmlns="http://www.w3schools.com «  
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  
  xsi:schemaLocation="Espace_TPXML base.xsd">
```

....

# XML Schema et grammaires d'arbre

Une grammaire d'arbre est un triplet (T,E,S,P) où

- E est un ensemble de noms d'éléments (terminaux)
- T est un ensemble de noms de types (non-terminaux)
- S est un sous-ensemble de E
- $R = \{t \rightarrow e \ r\}$  est un ensemble de règles telles que
  - $t \in T$  est un nom de type
  - $e \in E$  est un nom d'élément
  - r est une **expression régulières sur les noms de types**

DTD

- `<!ELEMENT officiel (#PCDATA | cinéma | film)*>`
- `<!ELEMENT cinéma (nom, adresse, (séance)*)>`

Grammaire d'arbres

- $T = \{T_{Offi}, T_{Cine}, T_{Film}, T_{Nom}, T_{Adresse}, T_{Séance}, PCdata\}$
- $E = \{officiel, cinéma, film, nom, adresse, séance\}$
- $S = \{T\}$

Règles P:

- $T_{Offi} \rightarrow officiel (T_{Cine} \mid T_{Film} \mid PCdata)^*$
- $T_{Cine} \rightarrow cinema (T_{Nom}, T_{Adresse}, T_{Séance}^*),$
- $T_{Film} \rightarrow film (Titre, Pays, Realisateur, Acteurs^*),$
- $T_{Nom} \rightarrow nom (Pddata), \dots \}$

DTD : une seule règle par élément

# XMLSchema : Grammaires à types uniques

---

## Types concurrents:

- Deux types différents A et B sont concurrents entre eux s'il existe deux règles de production  $A \rightarrow c r$  et  $B \rightarrow c r'$  pour le même élément c.

## XMLSchema est une grammaire d'arbres à types uniques

- les symboles dans S ne sont pas concurrents et,
- pour chaque règle de production, les non-terminaux dans son modèle de contenu ne sont pas concurrents.

## Validation simple:

- Dans une grammaire à types uniques il est possible d'identifier le type d'un élément à partir de son nom et du type de son parent.

# Conclusion : DTD et XML Schéma

---

DTD : grammaire d'arbre locale (chaque élément à un seul type)

XML Schema : grammaire d'arbre à type unique

- Plus expressif que DTD
- Validation reste simple

Il existe d'autres langages de typage XML :

- Relax NG, CDuce, etc.

Tous ces langages sont formellement fondés sur les grammaires/automates d'arbres.

