

Module MLBDA  
Master Informatique  
Spécialité DAC

Cours 5 – XSchema

# Contenu

- Xschema
  - Objectifs
  - Définition d'un schéma XML
  - Espaces de noms
  - Éléments et attributs
  - Typage et dérivation de types
  - Contraintes
  - Notion de clef et d'unicité

Documentation :

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

# Objectifs des schémas

- Définir un nouveau mécanisme de support à la modélisation
  - Reprenant les acquis des DTD (définition de modèles)
    - Arbres d'objets typés et valués
  - Permettant d'exprimer des contraintes fortes
    - Typage de données plus puissant et évolutif
  - Utilisant XML pour définir les modèles eux-mêmes (une seule syntaxe)
- Et aussi
  - Permettre de définir des contraintes incomplètes
    - Complémentarité aux DTD : outil de validation de données
  - Prendre en compte les espaces de noms (même nom, différents contextes)
  - Extensibilité
  - Syntaxe liée à un espace de noms (schéma bien formé, valide)
  - Notion de clef
  - Expression de la notion d'ensemble
  - Dérivation de types (restriction, extension)

# XMLschema

- Le schéma XML permet de rendre explicite certaines informations implicites de la DTD, par exemple
  - vérification de type
  - cardinalité des éléments
- Un schéma XML définit
  - les éléments
  - les attributs
  - les éléments fils, leur ordre, leur nombre,
  - le contenu des éléments
  - les types de données des éléments et attributs,
  - les valeurs par défaut et les valeurs fixes.

# Schéma XML

Document XML :

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

Une DTD pour ce document :

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

# Schéma XML

```
<?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)>

# 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** identifiés de façon unique, et on associe un nom (d'élément ou d'attribut) à l'espace dont il provient.
- Un espace de nom permet
  - 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 de noms d'éléments ou de noms d'attributs utilisables pour les éléments et les attributs d'un document XML.
- Chaque collection est identifiée par un URI (Unique Resource Identifier)
- Un nom  $n$  d'un espace de nom identifié par un URI  $u$  est désigné de façon unique par son nom étendu, qui est la paire  $(u, n)$ , où  $u$  est le nom de l'espace de nom, et  $n$  le **nom local**.
- A l'intérieur d'un élément XML :
  - Un préfixe est associé à chaque espace de nom,
  - Un nom d'un espace de noms est écrit sous forme d'un **nom qualifié** : *préfixe:nom local*



# Déclaration d'un espace de noms

- Pour utiliser les noms d'un espace de noms à l'intérieur d'un élément, on doit déclarer :
  - L'URI de cet espace
  - Un préfixe qui est un nom XML n'utilisant pas le caractère ':'
- Pour cela, on insère dans la balise ouvrante de l'élément, l'attribut :

**`xmlns:prefixe='URI de l'espace de noms'`**

- On peut déclarer un espace de noms par défaut avec l'attribut **`xmlns='URI de l'espace de noms'`**  
ou l'annihiler par la déclaration **`xmlns:=''`**

# Exemple

```
<schema xmlns= 'http://xyz.org/Xml-bib/biblio.dtd'>
```

définit l'URI associé à l'espace de noms par défaut.

```
<schema xmlns:math='http://www.w3.org/TR/REC-MathML.dtd'>
```

L'attribut **xmlns:math** associe le préfixe **math** à l'URI

```
'http://www.w3.org/TR/REC-MathML.dtd'
```

Le préfixe est un nom local, utilisé pour les attributs et les éléments, qui doit être déclaré par un ascendant.

# Visibilité d'une déclaration d'un espace de noms

- La déclaration d'un espace de noms muni d'un préfixe est visible dans l'élément la contenant et dans tous ses descendants sauf ceux pour lesquels un nouvel espace de même préfixe est déclaré.
- La déclaration d'un espace de noms par défaut est visible dans l'élément la contenant et dans tous ses descendants sauf ceux pour lesquels cette déclaration est annihilée ou ceux pour lesquels un nouvel espace de noms est déclaré.

# Association nom-espace de nom

- Tout nom d'élément ou nom d'attribut qui n'est pas une déclaration d'espace de noms, est un nom qualifié ayant l'une des deux formes suivantes :
  - *Prefixe:nom-local*
  - *Nom-local*
- Un nom qualifié préfixé appartient à l'espace de noms associé à ce préfixe déclaré dans le plus imbriqué des éléments contenant ce nom.
- Un nom qualifié non préfixé :
  - Appartient à l'espace de noms associé à ce préfixe déclaré dans le plus imbriqué des éléments contenant ce nom, s'il en existe un.
  - N'appartient pas à un espace de noms s'il n'existe pas de déclaration d'espace de noms par défaut dans les éléments contenant ce nom.

# Espaces de noms

- Deux types de collections de noms :
  - Vocabulaire pour définir un schéma :
    - element, attribute, schema, string, complexType, sequence, integer
  - Vocabulaire spécifique à un type de document :
    - note, pour, de, sujet, texte
- Un nom (namespace) est associé à chaque collection.
  - <http://www.w3.org/2001/XMLSchema> pour le vocabulaire de XMLSchema
  - <http://www.w3schools.com> pour le vocabulaire d'un document particulier.

# Elément schema

L'élément `<schema>` est l'élément racine de tout document XML Schema. Il peut contenir les attributs:

- **`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.).  
Doivent être préfixés par **`xs`**, et sont définis à l'adresse indiquée.
- **`targetNamespace="http://www.w3schools.com"`**  
indique l'espace de noms pour les éléments définis par ce schéma (note, de, pour, sujet, texte).

# Elément 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**.

Indique si les éléments ou attributs déclarés dans ce schéma doivent être qualifiés ou non (par l'espace nominal).

# Eléments (1)

- Un élément a un contenu, défini selon un modèle de contenu simple ou complexe
  - Modèle de contenu simple
    - Contenus typés : `string`, `boolean`, `float`, `timeInstant`, `timePeriod`, `month`, `date`, etc.).
  - 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"/>
```



# Occurrences

- On peut préciser le nombre d'occurrences d'un élément à l'aide des attributs `minOccurs` et `maxOccurs` (valeur par défaut : "1")

```
<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>
```

## Eléments (2)

- Contenu complexe (balise **complexType**)
  - Organisation des éléments entre eux (**sequence**, **choice**, **all**)
  - Contenus mixtes, avec des inter-relations entre éléments et caractères
  - Les éléments ayant des attributs sont de type complexe.
  - L'emboîtement des modèles de contenus définit une structure hiérarchique.

# Ordonnancement

- **Sequence** : séquence ordonnée d'éléments
- **Choice** : choix parmi un ensemble d'éléments
- **All** : séquence non ordonnée

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

# Eléments (3)

- Le contenu peut être défini de façon indépendante de l'élément, et référencé par l'attribut **ref**.
- Permet de définir des types simples ou complexes réutilisables
- 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>
```

# 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).

# Attributs

- Un attribut peut être restreint à un ensemble de valeurs

```
<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>
```

# Attributs

- Les déclarations d'attributs se font toujours après les déclarations d'éléments.
- Les déclarations d'attributs se font toujours par rapport à un élément (il n'y a pas d'attribut indépendant).

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

# Eléments et attributs

- Une information peut se déclarer sous forme d'éléments imbriqués ou à l'aide d'attributs.
- Ex :

```
<cours annee='2013' filiere='IAD'>MABD</cours>
```

```
<cours>
```

```
  <titre>MABD</nom>
```

```
  <annee>2013</annee>
```

```
  <filiere>IAD</filiere>
```

```
</cours>
```



# Typage

- Objectif : utiliser des **types** (simples ou complexes) **définis préalablement** dans des modèles de contenus ou des déclarations d'attributs.
- Un type peut être identifié et référencé par son **nom** (ou **anonyme**)
- Types simples prédéfinis : **string, integer, positiveInteger, negativeInteger, boolean, time, date, ID, IDREF, IDREFS**, etc.

(Attention : certains ne s'appliquent qu'aux attributs).

- Possibilité de créer de **nouveaux types**, basés sur des types prédéfinis
- Utilisation de **facettes** : permet de contraindre les valeurs
- Plusieurs facettes possibles sur un même type
- Possibilité de redéfinir un type à partir d'un autre

# Types simples et types complexes

- Un **type complexe** est un type qui peut contenir des déclarations d'éléments et d'attributs
- Un **type simple** ne peut pas contenir de déclarations d'éléments ou d'attributs.

# Types simples

- Un type simple est défini par
  - un ensemble de valeurs,
  - une représentation lexicale
  - un ensemble de facettes qui caractérise l'ensemble de valeurs
- Un type simple peut être
  - un type atomique
  - un type primitif ou dérivé
  - prédéfini ou défini par l'utilisateur

# Types prédéfinis ou définis par l'utilisateur

- Un type prédéfini peut être **primitif** ou **dérivé**
- Un type peut être défini par dérivation de types existants
- Un type défini peut être **anonyme** ou **nommé**

# Type simple

On utilise un type simple (élément **simpleType**) lorsqu'on veut créer un 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

- On utilise un type complexe (élément **complexType**) lorsqu'on veut définir des éléments fils et /ou des attributs pour un élément.
- Définition de modèles de contenus
  - ordonnancement d'éléments fils (séquence, choix)
  - Contenu mixte (attribut **mixed="true"**)
  - définition d'attributs
- Les types complexes peuvent contenir des définitions de types et des déclarations d'éléments

# Exemple (1)

```
<xs:complexType name="caType">
  <xs:sequence>
    <xs:element name="adresse" type="xs:string"
      minOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="datecreation" type="xs:date"/>
  <xs:attribute name="datemaj" type="xs:date"/>
</xs:complexType>

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

# Exemple (2)

Contenu mixte (mélange de texte et d'éléments)

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

Le titre pourra contenir les éléments **loc** et **heure** dans cet ordre, précédés et/ou suivi d'un contenu textuel.

Ex: *situation météo pour la région parisienne ce vendredi 15 novembre à 13h : temps frais et couvert.*

Par défaut, **mixed="false"**



# Type anonyme

```
<xs:element name="societe">  
  <xs:complexType>  
    <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>
```

# Type nommé

```
<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"/>
```

# Types anonymes et types nommés

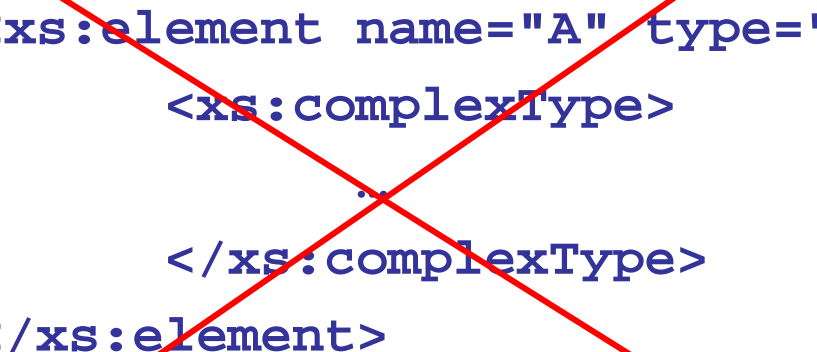
```
<xs:complexType name="foo">
  <xs:sequence>
    <xs:element name="B" .../>
    <xs:element name="C" .../>
  </xs:sequence>
</xs:complexType>
<xs:element name="A" type="foo"/>
```

Est équivalent à :

```
<xs:element name="A">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="B" .../>
      <xs:element name="C" .../>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Remarque

- Une déclaration d'élément peut avoir un attribut type ou un élément fils **complexType**, mais pas les deux simultanément!



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

# Dérivation de types

- 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
  - Permet d'ajouter des systèmes de contraintes
    - Ex: ajouter **maxOccurs= "1"** à un élément fils dans un type complexe
    - Ex: ajout d'une facette dans un type simple
- Dérivation par extension
  - Réutilisation d'un type complexe en ajoutant des informations supplémentaires (un attribut, un élément...)

# Restriction

```
<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>
```

# Restriction

Dans le type restreint, toutes les déclarations du type de base doivent être répétées, sauf dans le cas où un élément est déclaré avec minOccurs='0' et qu'on souhaite supprimer cet élément :

```
<xs:complexType name="Publication">
  <xs:sequence>
    <xs:element name="Titre" type="xs:string" maxOccurs="unbounded"/>
    <xs:element name="Auteur" type="xs:string" minOccurs="0"/>
  </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:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

La répétition est nécessaire à cause de la possibilité de substituer des éléments et/ou des types.

# Contraintes (Facettes)

Les **facettes** permettent de définir un nouveau type à partir d'un type existant (appelé le type de base), en spécifiant des valeurs pour une ou plusieurs facettes. C'est une façon d'exprimer des contraintes sur les éléments et les attributs.

Exemple :

Le type **string** a six facettes optionnelles :

- **length** : restreint les longueurs des types pour lesquels cela a un sens.
- **minLength**
- **maxLength**
- **pattern** : expression régulière définissant la suite de caractères acceptable. S'applique sur tous les types prédéfinis (sauf IDREFS, binary)
- **enumeration** : définit une liste de valeurs acceptables
- **whitespace** : spécifie comment les blancs sont gérés (tab, espace, retour chariot, etc.)



# Exemple (1)

```
<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 de données 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)

## Exemple (2)

```
<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 appelé forme. Un élément de ce type doit avoir comme valeur cercle, triangle ou carré.

# Facettes pour le type integer

Le type integer a 8 facettes optionnelles : `totalDigits`, `pattern`, `whitespace`, `enumeration`, `maxInclusive`, `minInclusive`, `maxExclusive`, `minExclusive`.

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

Crée un nouveau type Temperature. Les éléments de ce type sont des entiers, dont les valeurs sont entre -60 et +80 (valeurs incluses).

# Facettes

La forme générale pour spécifier un nouveau type à l'aide de facettes est :

```
<xs:simpleType name= "nom">
  <xs:restriction base= "xs:source">
    <xs:facet value= "value"/>
    <xs:facet value= "value"/>
    ...
  </xs:restriction>
</xs:simpleType>
```

La source est un type prédéfini.

Ex de facettes : `enumeration, length, minLength, maxLength, maxInclusive, minInclusive, maxExclusive, minExclusive, encoding, period, duration, pattern, whitespace, ...`

# 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)

```
<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>
```

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

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

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

# Dérivation d'un type simple

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

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

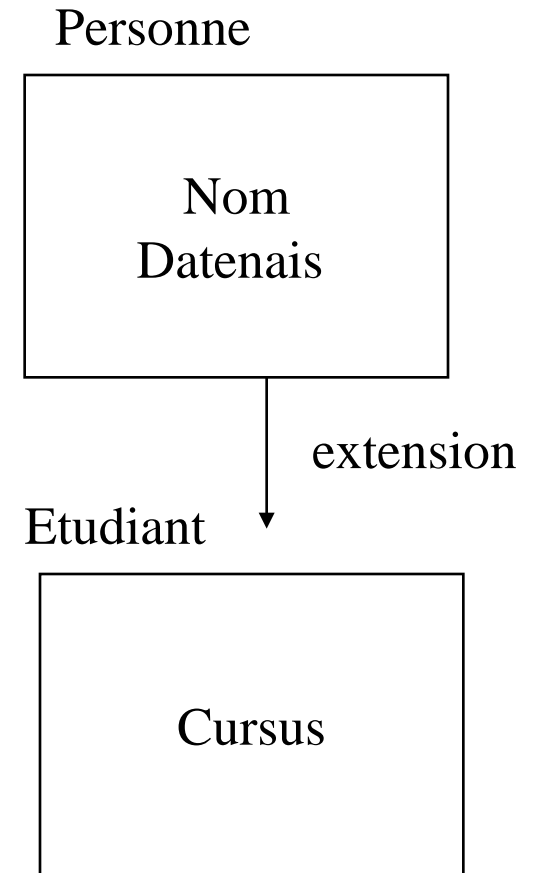
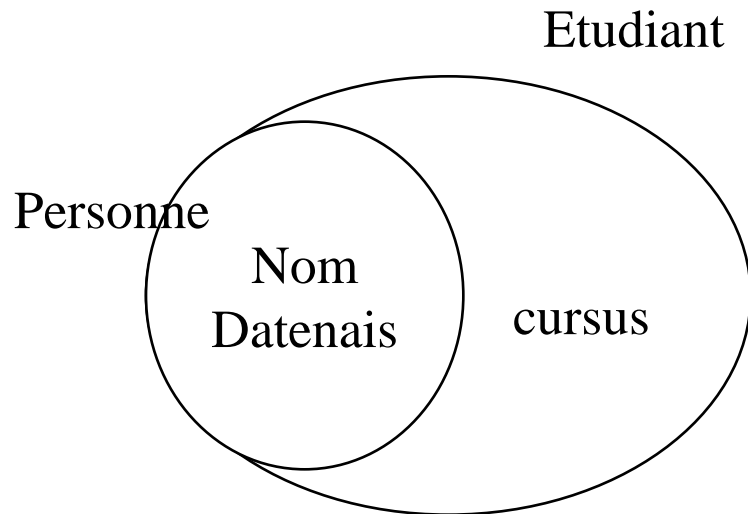
# Élément contenant un type simple

Deux façons équivalentes de déclarer un élément contenant un type simple :

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

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

# Dérivation par extension





# Extension

```
<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>
```

Les éléments de **etudiantType** ont 3 fils (**nom**, **datenais** et **cursus**).  
L'élément **cursus** est ajouté aux autres éléments **datenais** et **nom**.

# Types de contenu

- Concerne les types complexes
- Types de contenu par défaut :
  - Types simples : nœuds texte
  - Types 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>** ne contient rien d'autre que l'attribut **isbn**.

Ex: **<book isbn= "0123456789" />**

# Contenu simple (<simpleContent>)

Pour définir un élément ne contenant que du texte et des attributs, on dérive un type de données simple (ici le type string) en utilisant l'élément **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 épuisé
</book>
```

# Contenu complexe (<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"/>
```

# simpleContent vs complexContent

- **<simpleContent>** permet d'étendre ou de restreindre un type simple
- **<complexContent>** permet d'étendre ou de restreindre un type complexe.

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

X est un type complexe

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

Y est un type simple

# 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 **unique** et **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 (**nullable="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 **<key>** doit être imbriqué dans un élément, et doit se trouver après le modèle de contenu et les déclarations d'attributs.
- L'élément **<selector>** est utilisé comme un fils de **<key>** pour déterminer un ensemble d'éléments pour lesquels la clé s'applique (définit la portée du champ)
- L'élément **<field>** est utilisé comme un fils de **<key>** pour identifier l'élément ou l'attribut clé.
  - Il peut y avoir plusieurs éléments **<field>**.
- Les références doivent être qualifiées par le namespace, sauf si **elementFormDefault="unqualified"**



# Exemple

```
<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>
```

# Élément <key>

- Dans l'élément **Librairie**, on a défini une clef, appelée **Lclef** : on sélectionne (**selector**) chaque élément **Livre** de l'élément **Librairie**, et on définit (**field**) l'élément **ISBN** comme clef.
- Donc, dans l'élément **Librairie**, chaque élément **Livre** doit avoir un élément **ISBN**, et la valeur de cet élément doit être unique.
- Vérifié par le parser de schéma.

# Exemple

```
<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:element>

<xs:key name="Lclef">
  <xs:selector xpath="Livre"/>
  <xs:field xpath="Titre"/>
  <xs:field xpath="Auteur"/>
</xs:key>
</xs:element>
```

# Élément <unique>

- L'élément **<unique>** se définit de façon analogue à l'élément **<key>**. Il a un élément fils **<selector>** et un ou plusieurs éléments fils **<field>**.
- La seule différence est que l'existence de l'élément 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**.

# 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.
- Dans XML Schema, on définit un élément **<keyref>** signifiant que la valeur de cet élément doit correspondre à la valeur d'un élément référencé par **keyref**.

# Exemple

```
<Librairie>
  <LesLivres>
    <Livre>
      <Titre>Le soleil des Scorta</Titre>
      <Auteur>Laurent Gaudé</Auteur>
      <Date>2004</Date>
      <ISBN>0-123-45678-9</ISBN>      Élément <key>
      <Editeur>Actes Sud.</Editeur>
    </Livre>
    ...
  </LesLivres>
  <Signatures>
    <Auteur>
      <Nom>Laurent Gaudé</Nom>
      <LivreAsigner>
        <Titre>Le soleil des Scorta</Titre>
        <ISBN>0-123-45678-9</ISBN>      Élément <keyref>
      </LivreAsigner>
    </Auteur>
  </Signatures>
</Librairie>
```

# Exemple

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

```
<xs:keyref name="isbnRef" refer="Lclef">  
  <xs:selector xpath="Signatures/Auteur/LivreAsigner"/>  
  <xs:field xpath="ISBN"/>  
</xs:keyref>
```

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