المدرسة الدولية الخاصة للتقنيات
POLYTECH INTL

Le Standard XML
-Schéma XML -

Ichrak MEHREZ
(m_ichrak@hotmail.fr)

2018-2019

Rappel

- Le XML nous permet de créer notre propre vocabulaire grâce à un ensemble de règles et de balises personnalisables.
- Document XML valide: un document XML qui est *conforme aux règles syntaxiques*
- Les **fichiers de définition:** définir une structure stricte aux documents XML : les **DTD** et les **schémas XML**.

Introduction

- Les *schémas XML* permettent, comme les DTD, de définir des modèles de documents.
- Il est ensuite possible de vérifier qu'un document donné est valide pour un schéma, c'est-à-dire respecte les contraintes données par le schéma.
- Les schémas ont été introduits pour combler certaines lacunes des DTD.

Inconvénients DTD

- Les DTD manquent cruellement de précision dans la description des contenus des éléments.

  Exemple: Le seul type possible pour les contenus textuels est #PCDATA qui autorise toutes les chaînes de caractères (entier?, date?, heure?, …)

- Les DTD sont encore plus limitées dans la description des contenus mixtes
  La seule possibilité est de faire un mélange, **sans aucune contrainte**, de texte et de certains éléments.

- Déclaration de nouveaux types???

- Utilisation des éléments avec mêmes noms (types différents)???

Structure globale d'un schéma

- Un schéma XML se compose essentiellement de *déclarations d'éléments* et *d'attributs* et de *définitions de types*.

- Chaque élément est déclaré avec un type qui peut être, soit un des *types prédéfinis*, soit un nouveau *type défini* dans le schéma.

- Un nouveau type est obtenu soit par ***construction***, c'est-à-dire une description explicite des contenus qu'il autorise, soit par ***dérivation***, c'est-à-dire modification d'un autre type.

- Un schéma peut aussi contenir des imports d'autres schémas, des définitions de groupes d'éléments et d'attributs et des contraintes de cohérences.

Structure globale d'un schéma

- L'espace de noms des schémas XML est identifié par l'URI http://www.w3.org/2001/XMLSchema
- Il est généralement associé au préfixe xsd ou à xs.
- Tout le schéma est inclus dans l'élément xsd:schema.
- La structure globale d'un schéma est la suivante.

```
<?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>
```

DÉCLARTION D'ÉLÉMENTS

Déclaration

- Pour qu'un document soit valide pour un schéma, tout élément apparaissant dans le document doit être déclaré dans le schéma.
- Cette déclaration lui donne un type qui détermine, d'une part, les contenus possibles et, d'autre part, les attributs autorisés et obligatoires.
- Contrairement aux DTD, les attributs ne sont pas directement associés aux éléments. Ils font partie des types qui sont donnés aux éléments.
- Le type donné à un élément peut être soit un type nommé soit un type anonyme.

Type nommé

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

- *element* et *type* sont respectivement le nom et le type de l'élément.

- Ce type peut être un des types prédéfinis comme xsd:string ou xsd:integer ou encore un type défini dans le schéma

Valeur par défaut et valeur fixe

- Lorsque le type est simple, il est possible de donner une valeur par défaut ou une valeur fixe à l'élément .

- Il faut pour cela donner des valeurs aux attributs *default* ou *fixed* de l'élément *xsd:element*.

```
<xsd:element name="title" type="xsd:string" default="Titre par défaut"/>
```

```
<xsd:element name="title" type="xsd:string" fixed="Titre fixe"/>
```

Type anonyme

- Lors de la déclaration d'un élément, il est possible de décrire explicitement le type. La déclaration du type est alors le contenu de l'élément *xsd:element*. Le type est alors local et sa déclaration prend alors une des deux formes suivantes où *element* est le nom de l'élément déclaré.

```
<xsd:element name="element">
    <xsd:simpleType>
    ...
    </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="element">
    <xsd:complexType>
    ...
    </xsd:complexType>
</xsd:element>
```

Référence à un élément global

- Un élément global, c'est-à-dire dont la déclaration par *xsd:element* est enfant direct de l'élément *xsd:schema*, peut être utilisé par des définitions de type. Ces définitions de type se contentent de faire référence à l'élément global de la façon suivante.

```
<!-- Déclaration globale de l'élément title -->
<xsd:element name="title" type="Title"/>
...
<!-- Définition d'un type global ou local -->
<xsd:complexType ... >
...
<!-- Utilisation de l'élément title -->
<xsd:element ref="title"/>
...
</xsd:complexType>
```

Éléments locaux

- Deux éléments définis non globalement dans un schéma peuvent avoir le même nom tout en ayant des types différents. Il s'agit en fait d'éléments différents mais ayant le même nom.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    ...
    <xsd:element name="strings">
        <xsd:complexType>
            <xsd:sequence>
            <!-- Déclaration du premier élément local -->
            <xsd:element name="local" type="xsd:string" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="integers">
        <xsd:complexType>
            <xsd:sequence>
            <!-- Déclaration du second élément local -->
            <xsd:element name="local" type="xsd:integer" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

```
<lists>
    <strings>
        <local>Une chaîne</local>
        <local>A string</local>
    </strings>
    <integers>
        <local>-1</local>
        <local>1</local>
    </integers>
</lists>
```

DÉFINITIONS DES TYPES

Définition

- Les schémas XML distinguent les *types simples* introduits par le constructeur **xsd:simpleType** et les *types complexes* introduits par le constructeur **xsd:complexType**.

- Les types simples
  - Décrivent des contenus textuels, c'est-à-dire ne contenant que du texte.
  - Peuvent être utilisés pour les éléments comme pour les attributs
  - Généralement obtenus par dérivation des types prédéfinis.

- Les types complexes
  - Décrivent des contenus purs constitués uniquement d'éléments ou des contenus mixtes constitués de texte et d'éléments.
  - Peuvent uniquement être utilisés pour déclarer des éléments.

Définition

- Les schémas permettent de définir une hiérarchie de types qui sont obtenus par *extension* ou *restriction* de types déjà définis.
- L'extension de type est similaire à l'héritage des langages de programmation orientés objet. Elle permet de définir un nouveau type en ajoutant des éléments et/ou des attributs à un type.
- La restriction permet au contraire d'imposer des contraintes supplémentaires au contenu et aux attributs.
- Tous les types prédéfinis ou définis dans un schéma sont dérivés du type xsd:anyType. Type par défaut lorsqu'une déclaration d'élément ne spécifie pas le type comme la déclaration suivante.

```
<!-- Le type de l'élément object est xsd:anyType -->
<xsd:element name="object"/>
```

Types prédéfinis

- *Types numériques (1/2)*

xsd:integer	Nombre entier sans limite de précision. Ce type n'est pas primitif et dérive du type xsd:decimal.
xsd:positiveInteger	Nombre entier strictement positif sans limite de précision
xsd:negativeInteger	Nombre entier strictement négatif sans limite de précision
xsd:nonPositiveInteger	Nombre entier négatif ou nul sans limite de précision
xsd:nonNegativeInteger	Nombre entier positif ou nul sans limite de précision
xsd:float	Nombre flottant sur 32 bits
xsd:double	Nombre flottant sur 64 bits
xsd:decimal	Nombre décimal sans limite de précision

Types prédéfinis

- *Types numériques (2/2)*

xsd:boolean	Valeur booléenne avec true ou 1 pour vrai et false ou 0 pour faux
xsd:byte	Nombre entier signé sur 8 bits
xsd:unsignedByte	Nombre entier non signé sur 8 bits
xsd:short	Nombre entier signé sur 16 bits
xsd:unsignedShort	Nombre entier non signé sur 16 bits
xsd:int	Nombre entier signé sur 32 bits
xsd:unsignedInt	Nombre entier non signé sur 32 bits
xsd:long	Nombre entier signé sur 64 bits. Ce type dérive du type xsd:integer.
xsd:unsignedLong	Nombre entier non signé sur 64 bits

Types prédéfinis

- *Types pour les chaînes et les noms*

xsd:string	Chaîne de caractères composée de caractères Unicode
xsd:normalizedString	Chaîne de caractères normalisée, c'est-à-dire ne contenant pas de tabulation, de saut de ligne ou de retour chariot
xsd:token	Chaîne de caractères normalisée (comme ci-dessus) et ne contenant pas en outre des espaces en début ou en fin ou des espaces consécutifs
xsd:Name	Nom XML
xsd:QName	Nom qualifié
xsd:NCName	Nom non qualifié, c'est-à-dire sans caractère ':'
xsd:language	Code de langue sur deux lettres de la norme ISO 639 comme fr ou en éventuellement suivi d'un code de pays de la norme ISO 3166 comme en-GB.
xsd:anyURI	Un URI
xsd:base64Binary	Données binaires représentées par une chaîne au format Base 64.
xsd:hexBinary	Données binaires représentées par une chaîne au format Hex.

Types prédéfinis

- *Types pour les dates et les heures*

xsd:time	Heure au format hh:mm:ss[.sss][TZ]
xsd:date	Date au format YYYY-MM-DD
xsd:dateTime	Date et heure au format YYYY-MM-DDThh:mm:ss
xsd:duration	Durée au format PnYnMnDTnHnMnS comme P1Y6M, P1M12DT2H
xsd:dayTimeDuration	Durée au format PnDTnHnMnS comme P7DT4H3M2S.
xsd:yearMonthDuration	Durée au format PnYnM comme P1Y6M.
xsd:gYear	Année du calendrier grégorien au format YYYY comme 2011.
xsd:gYearMonth	Année et mois du calendrier grégorien au format YYYY-MM
xsd:gMonth	Mois du calendrier grégorien au format MM comme 01 pour janvier.
xsd:gMonthDay	Jour et mois du calendrier grégorien au format MM-DD comme 12-25 pour le jour de Noël.
xsd:gDay	Jour (dans le mois) du calendrier grégorien au format DD comme 01 pour le premier de chaque mois.

Types prédéfinis

- *Types hérités des DTD*

xsd:ID	nom XML identifiant un élément
xsd:IDREF	référence à un élément par son identifiant
xsd:IDREFS	liste de références à des éléments par leurs identifiants
xsd:NMTOKEN	jeton
xsd:NMTOKENS	liste de jetons séparés par des espaces
xsd:ENTITY	entité externe non XML
xsd:ENTITIES	liste d'entités externes non XML séparées par des espaces
xsd:NOTATION	notation

Types simples

- Les types simples définissent uniquement des contenus textuels.
- Ils peuvent être utilisé pour les éléments ou les attributs.
- Ils sont introduits par l'élément xsd:simpleType.
- Un type simple est souvent obtenu *par restriction* d'un autre type défini. Il peut aussi être construit *par union* d'autres types simples ou *par l'opérateur de listes*.

```
<xsd:simpleType ...>
...
</xsd:simpleType>
```

Types simples

- L'élément xsd:simpleType peut avoir un attribut *name* si la déclaration est globale.

- La déclaration du type se fait ensuite dans le contenu de l'élément xsd:simpleType comme dans l'exemple suivant.

```
<xs:simpleType name="age">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Types complexes

- Les types complexes définissent des contenus purs (constitués uniquement d'éléments), des contenus textuels ou des contenus mixtes.
- Tous ces contenus peuvent comprendre des attributs.
- Les types complexes peuvent seulement être utilisés pour les éléments.
- Ils sont introduits par l'élément **xsd:complexType**.

Types complexes

- Un type complexe peut être construit *explicitement* ou être dérivé d'un autre type par *extension* ou *restriction*.
- La construction explicite d'un type se fait en utilisant les opérateurs de séquence xsd:sequence, de choix xsd:choice ou d'ensemble xsd:all.
- La construction du type se fait directement dans le contenu de l'élément xsd:complexType.

```
<!-- Type explicite -->
<xsd:complexType ...>
<!-- Construction du type avec xsd:sequence, xsd:choice ou xsd:all -->
...
</xsd:complexType>
```

Types complexes

- Si le type est obtenu par extension ou restriction d'un autre type, l'élément xsd:complexType doit contenir un élément **xsd:simpleContent** ou **xsd:complexContent** qui précise si le contenu est purement textuel ou non.

```
<!-- Type dérivé à contenu textuel -->
<xsd:complexType ...>
<xsd:simpleContent>
<!-- Extension ou restriction -->
...
</xsd:simpleContent>
</xsd:complexType>
```

```
<!-- Type dérivé à contenu pur ou mixte -->
<xsd:complexType ...>
<xsd:complexContent>
<!-- Extension ou restriction -->
...
</xsd:complexContent>
</xsd:complexType>
```

Contenu mixte

- L'attribut ***mixed*** de l'élément xsd:complexType permet de construire un type avec du contenu mixte. Il faut, pour cela, lui donner la valeur ***true***.

```
<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>
```

```
<personne>
    Nom : <nom>Tounsi</nom>,
    Prénom : <prenom>Ali</prenom>.
</personne>
```

CONSTRUCTION DE TYPES

Élément vide

- Si un type complexe ***déclare uniquement des attributs***, le contenu de l'élément doit être vide.
- Par exemple, le type suivant déclare un type *Link*. Tout élément de ce type doit avoir un contenu vide et un attribut *ref* de type *xsd:IDREF*. Il s'agit en fait d'une extension du type vide par ajout d'attributs.

```
<xsd:element name="link" type="Link"/>
<xsd:complexType name="Link">
    <xsd:attribute name="ref" type="xsd:IDREF" use="required"/>
</xsd:complexType>
```

```
<link ref="id-42"/>
```

Opérateur de séquence

- L'opérateur **xsd:sequence** définit un nouveau type formé d'une suite des éléments énumérés. C'est l'équivalent de l'opérateur ',' des DTD

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

```
<personne>
  <cin>01234567</cin>
  <nom>Tounsi</nom>
  <prenom>Ali</prenom>
</personne>
```

- Le nombre d'occurences de chaque élément dans la séquence est 1 par défaut mais il peut être modifié par les attributs *minOccurs* et *maxOccurs* *(A voir dans la suite …)*

Opérateur de choix

- L'opérateur **xsd:choice** définit un nouveau type formé d'un des éléments énumérés. C'est l'équivalent de l'opérateur '|' des DTD

```
<xsd:element name="publication">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref="book"/>
            <xsd:element ref="article"/>
            <xsd:element ref="report"/>
            </xsd:choice>
    </xsd:complexType>
</xsd:element>
```

- Il est bien sûr possible d'imbriquer les opérateurs xsd:sequence et xsd:choice.

Opérateur d'ensemble

- L'opérateur **xsd:all** n'a pas d'équivalent dans les DTD. Il définit un nouveau type dont chacun des éléments doit apparaître une fois dans un ordre quelconque

```
<xsd:element name="book">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="title" type="xsd:string"/>
            <xsd:element name="author" type="xsd:string"/>
            <xsd:element name="year" type="xsd:string"/>
            <xsd:element name="publisher" type="xsd:string"/>
        </xsd:all>
    </xsd:complexType>
</xsd:element>
```

Opérateur d'ensemble

- xsd:all ne peut pas être imbriqué avec d'autres constructeurs xsd:sequence, xsd:choice ou même xsd:all.
- Les seuls enfants possibles de **xsd:all** sont des éléments xsd:element.
- xsd:all est toujours enfant de xsd:complexType ou xsd:complexContent.
- Les attributs minOccurs et maxOccurs des éléments apparaissant sous l'opérateur xsd:all
  - minOccurs doit être 0 ou 1 et la valeur de l'attribut
  - maxOccurs doit être 1 qui est la valeur par défaut.
- Les attributs minOccurs et maxOccurs peuvent aussi apparaître comme attribut de xsd:all.
  - Leurs valeurs s'appliquent à tous les éléments enfants de xsd:all.
  - Les valeurs autorisées pour minOccurs sont 0 et 1 et la seule valeur autorisée pour maxOccurs est 1.

Opérateur d'union

- L'opérateur xsd:union définit un nouveau type simple dont les valeurs sont celles des types listés dans l'attribut ***memberTypes***.

```
<xsd:simpleType name="Unbounded">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="unbounded"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="IntegerOrUnbounded">
    <xsd:union memberTypes="xsd:nonNegativeInteger Unbounded"/>
</xsd:simpleType>
```

- Nouveau type *IntegerOrUnbounded:* Les valeurs de ce type sont soit un entier positif ou nul du type *xsd:nonNegativeInteger* soit l'unique chaîne « *unbounded* » du type *Unbounded*.
- Ce dernier type est obtenu par restriction du type *xsd:string*.

Opérateur de liste

- **xsd:list** définit un nouveau type simple dont les valeurs sont les listes de valeurs du type simple donné par l'attribut **itemType**.
- Il s'agit uniquement de listes de valeurs séparées par des espaces.
- L'exemple suivant définit des types pour les listes d'entiers et pour les listes de 5 entiers.

```
<!-- Type pour les listes d'entiers -->
<xsd:simpleType name="IntList">
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>

<!-- Type pour les listes de 5 entiers -->
<xsd:simpleType name="IntList5">
    <xsd:restriction base="IntList">
        <xsd:length value="5"/>
    </xsd:restriction>
</xsd:simpleType>
```

Répétitions

- Les attributs **minOccurs** et ***maxOccurs*** permettent de préciser le nombre minimal ou maximal d'occurrences d'un élément ou d'un groupe.
- Ils sont l'équivalent des opérateurs **?**, ***** et **+** des DTD.
- Ils peuvent apparaître comme attribut des éléments *xsd:element*, *xsd:sequence*, *xsd:choice* et *xsd:all*.
- 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:any

- **xsd:any** permet d'introduire dans un document un ou des éléments externes au schéma, c'est-à dire non définis dans le schéma.
- Le nombre d'éléments externes autorisés peut-être spécifié avec les attributs *minOccurs* et *maxOccurs*.
- La validation de ces éléments externes est contrôlée par l'attribut ***processContents*** qui peut prendre les valeurs ***strict***, ***lax*** et ***skip***(La valeur par défaut est strict).
  - ***strict:*** les éléments externes doivent être validés par un autre schéma déterminé par l'espace de noms de ces éléments pour que le document global soit valide.
  - ***skip***: les éléments externes ne sont pas validés.
  - ***lax:*** est intermédiaire entre strict et skip. La validation des éléments externes est tentée mais elle peut échouer.

xsd:any

- Le schéma suivant autorise zéro ou un élément externe dans le contenu de l'élément *person* après l'élément *lastname*.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="person">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="firstname" type="xsd:string"/>
                <xsd:element name="lastname" type="xsd:string"/>
                <xsd:any processContents="lax" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0" encoding="utf-8"?>
<person>
    <firstname>Elizabeth II Alexandra Mary</firstname>
    <lastname>Windsor</lastname>
    <title>Queen of England</title>
</person>
```

EXERCICES

LES ATTRIBUTS

Déclaration

- La déclaration d'un attribut utilise l'élément ***xsd:attribute***

```
<xsd:attribute name="name" type="type"/>
```

- Les attributs ***name*** et ***type*** de ***xsd:attribute*** spécifient respectivement le nom et le type de l'attribut.

- Le type d'un attribut est ***nécessairement un type simple*** puisque les attributs ne peuvent contenir que du texte.

- Exemple

```
<xsd:attribute name="format" type="xsd:string"/>
```

Déclaration

- Comme pour un élément, le type d'un attribut peut être ***anonyme***. Il est alors défini dans le contenu de l'élément **xsd:attribute**.

- Exemple: la valeur de l'attribut *lang* déclaré ci-dessous peut être la chaîne *en* ou la chaîne *fr*.

```
<xsd:attribute name="lang">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="en"/>
            <xsd:enumeration value="fr"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```

Déclaration

- Les déclarations d'attributs se placent normalement dans les définitions de types complexes qui peuvent être globaux ou locaux.
- Les types simples ne peuvent pas avoir d'attributs.
- La définition d'un type complexe se compose de la description du contenu suivie de la déclaration des attributs.
- L'ordre de déclarations des attributs est sans importance puisque l'ordre des attributs dans une balise n'est pas fixe.

Déclaration

- Exemple

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Type global et simple Lang pour l'attribut lang -->
    <xsd:simpleType name="Lang">
        <xsd:restriction base="xsd:string">
        <xsd:length value="2"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="List">
        <!-- Contenu du type List -->
        <xsd:sequence maxOccurs="unbounded">
            <xsd:element name="item" type="xsd:string"/>
        </xsd:sequence>
        <!-- Déclaration des attributs locaux form et lang du type List -->
        <xsd:attribute name="form" type="xsd:string"/>
        <xsd:attribute name="lang" type="Lang"/>
    </xsd:complexType>

    <xsd:element name="list" type="List"/>
</xsd:schema>
```

Déclaration

- Un attribut peut aussi être global lorsque sa déclaration par ***xsd:attribute*** est enfant direct de l'élément ***xsd:schema***.

- Cet attribut peut alors être ajouté à différents types complexes.

- La définition du type utilise l'élément ***xsd:attribute*** avec un attribut ***ref*** qui remplace les deux attributs ***name*** et ***type***. Cet attribut ***ref*** contient le nom de l'attribut global à ajouter.

- La déclaration globale d'un attribut est justifiée lorsque celui-ci a des occurrences multiples. Elle accroît la modularité des schémas en évitant de répéter la même déclaration dans plusieurs types

Déclaration

- Exemple:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Déclaration de l'attribut global lang -->
<xsd:attribute name="lang" type="xsd:language"/>
<xsd:element name="texts" type="Texts"/>
    <xsd:complexType name="Texts">
        <xsd:sequence>
            <xsd:element name="text" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                            <!-- Ajout de l'attribut lang au type anonyme -->
                            <xsd:attribute ref="lang"/>
                        </xsd:extension>
                    </xsd:simpleContent>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <!-- Ajout de l'attribut lang au type Texts -->
        <xsd:attribute ref="lang"/>
    </xsd:complexType>
</xsd:schema>
```

Attribut optionnel, obligatoire ou interdit

- Par défaut, un attribut est optionnel. Il peut aussi être rendu *obligatoire* ou *interdit* en donnant la valeur ***required*** ou ***prohibited*** à l'attribut ***use*** de l'élément ***xsd:attribute***.

```
<xsd:attribute name="lang" type="xsd:NMTOKEN" use="optional"/>
<xsd:attribute name="xml:id" type="xsd:ID" use="required"/>
<xsd:attribute name="dummy" type="xsd:string" use="prohibited"/>
```

- L'attribut ***use*** peut aussi prendre la valeur ***optional***. Cette valeur est très peu utilisée car c'est la valeur par défaut.

Valeur par défaut et valeur fixe

- Il est possible de donner une valeur par défaut ou une valeur fixe à un attribut via l'attribut *default* ou de l'attribut ***fixed*** de l'élément xsd:attribute.

- Une valeur par défaut n'est autorisée que si l'attribut est optionnel.

- Il est également interdit de donner simultanément une valeur par défaut et une valeur fixe.

```
<xsd:attribute name="lang" type="xsd:NMTOKEN" default="fr"/>
<xsd:attribute name="lang" type="xsd:NMTOKEN" fixed="en"/>
```

Faire référence à un schéma

- On utilise un des attributs *schemaLocation* ou *noNamespaceSchemaLocation* dans **l'élément racine** du document à valider.

- Ces deux attributs se trouvent dans l'espace de noms des instances de schémas identifié par l'URI http://www.w3.org/2001/XMLSchema-instance.

- L'attribut *schemaLocation* est utilisé lors de l'utilisation d'espaces de noms alors que l'attribut *noNamespaceSchemaLocation* est utilisé lorsque le document n'utilise pas d'espace de noms.

Faire référence à un schéma

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="personne">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="cin" type="xs:string" />
          <xs:element name="nom" type="xs:string" />
          <xs:element name="prenom" type="xs:string" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<personne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="personne.xsd">
  <cin>01234567</cin>
  <nom>Tounsi</nom>
  <prenom>Ali</prenom>
</personne>
```

Faire référence à un schéma

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://example.com/ns"
            elementFormDefault="qualified">
  <xsd:element name="personne">
      <xsd:complexType>
        <xsd:sequence>
         <xsd:element name="cin" type="xsd:string" />
          <xsd:element name="nom" type="xsd:string" />
          <xsd:element name="prenom" type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns:personne xmlns:ns="http://example.com/ns"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://example.com/ns ns.xsd">
  <ns:cin></ns:cin>
  <ns:nom></ns:nom>
  <ns:prenom></ns:prenom>
</ns:personne>
```