Extensible Markup Language (XML)

Ahmad Mazyad

LISIC - Université du Littoral Côte d'Opale

2018

Outline

Support de Cours

- Beginning XML 5 edition (JOE FAWCETT, . . .)
  Un trés bon cours pour commencer mais en anglais
- L'essentiel de XML: Cours XML (Olivier Carton)
  Support du cours XML en M2 Pro à l'Université Paris Diderot
  https://www.irif.fr/ carton/Enseignement/XML/Cours/support.pdf
- Structurez vos données avec XML
  Un cours en ligne, simple, facile à suivre, mais . . .
  https://openclassrooms.com/courses/structurez-vos-donnees-avec-xml/qu-est-ce-que-le-xml
- Attributes or Elements
  https://www.ibm.com/developerworks/library/x-eleatt/
- slides: goo.gl/yPWqm6

What Is XML?

- Extensible Markup Language (Langage de balisage generique).
- Décrire et structurer des données selon certaines contraintes définies.

History

- SGML (Standard Generlized Markup Language)
- HTML - 1991
- XML - First draft en 1996
- XML - Première version publiée par W3C en 1998
- XHTML - Redéfinition du HTML 4.0 en 1999

XHTML

- HTML version XML (non utilisé depuis HTML 5)

XML vs HTML

- HTML décrit la présentation et XML décrit le contenu.
- HTML est une application de SGML et XML est “un sous-ensemble de SGML”

XML vs JSON

- JSON est un format de données.
- XML est une langage de balisage.

Applications XML

Configuration files, Web services, Web Content (XHTML), Document Management, Database Systems, Image Representatipon (SVG)

Syntaxe de XML

Example

```
1  <?xml version="1.0" encoding="UTF-8"?>①
2  <!-- Time-stamp: "bibliography.xml 3 Mar 2008" -->②
3  <!DOCTYPE bibliography SYSTEM "bibliography.dtd" >③
4  <bibliography>④
5     <book key="Michard01" lang="fr">⑤
6        <title>XML langage et applications</title>
7        <author>Alain Michard</author>
8        <year>2001</year>
9        <publisher>Eyrolles</publisher>
10       <isbn>2-212-09206-7</isbn>
11       <url>http://www.editions-eyrolles/livres/michard/</url>
12       <image href="michard.png" />⑥
13    </book>
14    <book key="Zeldman03" lang="en">
15       <title>Designing with web standards</title>
16       ...
17    </book>
18 </bibliography>⑦
```

Explication

① Entête **XML** avec la version *1.0* et l'encodage *UTF-8* des caractères.
② Commentaire délimité par les chaînes de caractères $<! --$ et $-- >$.
③ Déclaration de **DTD** externe dans le fichier *bibliography.dtd*.
④ Balise **ouvrante** de l'élément racine *bibliography*
⑤ Balise ouvrante de l'élément *book* avec deux attributs de noms *key* et *lang* et de valeurs *Michard01* et *fr*
⑥ Balise **auto fermante** <image / >
⑦ Balise **fermante** de l'élément racine *bibliography*

Syntaxe de XML

Prologue

```
1 Entête
2 <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
3 Commentaire
4 <!-- Time-stamp: "bibliography.xml 3 Mar 2008" -->
5 DOCTYPE
6 <!DOCTYPE bibliography SYSTEM "bibliography.dtd" >
```

Entête

- Première partie d'un document XML.
- L'attribut *version* précise la version de XML utilisée (*1.0*, *1.1*). (**Obligatoire**)
- *encoding* indique le jeu de caractères utilisé (**UTF-8**, US-ASCII, ISO-8859-1, et UTF-16).
- *standalone* permet d'indiquer si le document est autonome ou rattaché à un fichier DTD (**no**, yes).

Corps du Document

```
<element [attribut="valeur" autre_attribut="valeur2"]>
   [<element_auto_fermante [attribut="valuer"] />
   <element_imbriuque>[data]</element_imbriuque>
   data]
</element>
```

Elément

- Formé d'une balise ouvrante, d'un contenu et d'une balise fermante.
- Quand le contenu est vide, on peut utiliser la balise auto fermante $< balise/ >$.
- Un élément peut avoir zero, un ou plusieurs attributs uniques.

Imbrication des Eléments

$< balise1 >< balise2 >< /balise1 >< /balise2 >$ (Invalide)
Tous les documents doivent avoir un seul élément racine.

Corps du Document

```
<element [attribut="valeur" autre_attribut="valeur2"]>
   [<element_auto_fermante [attribut="valuer"] />
   <element_imbriuque>[data]</element_imbriuque>
   data]
</element>
```

Attribut

- Un attribut est constitué d'un nom et d'une valeur separés par un signe égal $=$.
- La valeur de l'attribut doit être entre guillemets simples ou doubles.
- Un attribut doit être unique pour un élément.

Corps du Document

Attributs particuliers

- xml:lang décrit la langue du contenu de l'élément
  Bonjour
- xml:space permet d'indiquer à une application le traitement des caractères d'espacement (**default** et preserve).
- xml:base permet de préciser l'URI de base d'un élément.
  <chapter xml:base="XML/chapter.html">
- xml:id permet d'associer un identificateur à tout élément indépendamment de toute DTD ou de tout schéma.

Corps du Document

Style de Nomenclature (Naming Style)

- Pascal-casing: Ceci met en majuscules les mots séparés, y compris le premier: $< MyElement / >$.
- Camel-casing: Similaire à Pascal sauf que la première lettre est en minuscule: $< myElement / >$.
- Mots séparés par un trait de souligement: $< my\_element / >$

Spécifications de Nomenclature

Règles à suivre pour nommer les éléments et les attributs.

- le nom d'un élément/Attribut commence avec [a-zA-Z_].
- Les caractères suivants peuvent également être un tiret (-) ou un chiffre.
- Les noms étant sensibles à la casse, les balises de début et de fin doivent correspondre exactement.
- Ne pas utiliser XML au début d'un nom.

Corps du Document

```
<conversionData>
   1 kilometer &lt; 1 mile
   1 pound &lt; 1 kilogram
</conversionData>
<conversionData><![CDATA[
   1 kilometer < 1 mile
   1 pound < 1 kilogram
]]></conversionData>
```

Sections Littérales/CDATA

Permet d'utiliser les caractères spéciaux ($<$) directement au lieu de des Entités prédéfinies (&*lt*;).

Quand utiliser les éléments et quand les attributs?

```
<applicationUsers>
   <user firstName="Joe" lastName="Fawcett" />
   <user firstName="Danny" lastName="Ayers" />
</applicationUsers>

<applicationUsers>
   <user>
      <firstName>Joe</firstName>
      <lastName>Fawcett</lastName>
   </user>
   <user>
      <firstName>Danny</firstName>
      <lastName>Ayers</lastName>
   </user>
</applicationUsers>
```

```
<book xmlns:xi="http://www.w3.org/2001/XInclude">
...
<!-- Inclusion des differents chapitres -->
   <xi:include href="introduction.xml" parse="xml"/>
   <xi:include href="Syntax/chapter.xml" parse="xml"/>
...
</book>
-----------------------------------------------
<xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="file.php?id=1&c=generatexmlfile" />
```

XInclude

répartir un gros document en plusieurs fichiers afin d'en rendre la gestion plus aisée.

Excercice 1

Il vous est demandé de réaliser un format XML permettant de stocker des recettes de cuisine et gérant également les stocks des ingrédients que vous possédez. Chaque recette de cuisine a un nom, une description, la durée de préparation et la durée de cuisson, le nombre de calories par personne, le nombre de parts et le niveau de difficulté : difficile, moyen ou facile. Pour chaque recette vous voulez savoir quels sont les ingrédients nécessaires et la quantité associée à chaque ingrédient. Pour chaque ingrédient vous avez son nom et le nombre de calories pour 100 grammes de cet ingrédient. Un même ingrédient peut avoir plusieurs conditionnements, par exemple, l'ingrédient farine peut être stocké sous forme d'un paquet de 1 kg ou de 500 g. Pour gérer les stocks des ingrédients, le lieu de stockage de produits(conditionnement) est mémorisé.

Question: Proposer un format XML.

Exercice 2

L'objectif de l'exercice est de proposer un format XML permettant de representer les etapes ainsi que les informations de chaque étape de la fabrication du champage.

- Le pressurage consiste à presser manuellement le raisin pour en faire éclater les baies.
- La fermentation est la phase de mise en cuve du jus de raisin. Celle-ci dure quinze jours environ. Conservée à une température constante de 18-20 C, cette 1ère fermentation dite "alcoolique" active les levures naturelles présentes dans le jus de raisin et transforme les sucres en un mélange d'alcool et de gaz carbonique.
- La clarification débarrassera le vin des levures ou autres particules solides qui altèrent sa saveur.
- Le vigneron réalise l'assemblage en mélangeant des vins "tranquilles" (non effervescents) issus de différentes récoltes pour une saveur finale constante. Cette étape constitue la véritable "signature" d'une cuvée de champagne.
- Le vin obtenu est mis en bouteille, additionné de sucre et levures. Cette seconde fermentation le transforme en vin effervescent. L'étape du tirage permet la "prise de mousse", la bouteille est bouchée d'un "bidule" (capsule).
- Le vin reposera 15 mois pour la fabrication d'un champagne brut et 36 mois pour obtenir un Millésime.
- Le remuage, manuel ou mécanique, consiste à faire tourner la bouteille de gauche à droite, puis à lui mettre la tête en bas, pour faciliter l'accumulation des dépôts organiques.
- Ceux-ci seront gelés puis évacués naturellement lors du dégorgement effectué par le col de la bouteille plongé dans un liquide à -25C. La bouteille est alors prête à recevoir le bouchon de liège, la capsule, le muselet, l'étiquette et la collerette qui finalisent la fabrication du champagne.

Question: Proposer un format xml.

Espaces de noms (Namespaces)

Espaces de noms?

Les espaces de noms sont un moyen de regouper les éléments et les attributs sous un en-tête commun afin de les différencier des éléments portant le même nom.

Choisir un Nom?

N'importe quelle chaîne de caractères peut être utilisé.
W3C recommande d'utiliser les URIs (URL, URN).
e.g. http://univ-littoral.fr/namespaces/m1/xml

En Pratique

Les espaces de noms sont surtout utilisés dans ces cas:

- XML Schemas: définir la structure d'un document
- Combinaison des documents de plusieurs source.
- Versioning

Espaces de noms

Example

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <book version="5.0" xml:lang="fr"
       xmlns="http://docbook.org/ns/docbook"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:xi="http://www.w3.org/2001/XInclude">
3     <title>Langages formels</title>
4     <!-- MEtadonnEes -->
5     <metadata>
6        <dc:title>Langages formels</dc:title>
7        <dc:creator>Olivier Carton</dc:creator>
8        <dc:date>2008-10-01</dc:date>
9     </metadata>
10    <!-- Import des chapitres avec XInclude -->
11    <xi:include href="introduction.xml" parse="xml" />
12    <xi:include href="chapter1.xml" parse="xml" />
13 </book>
```

Déclarer un espace de noms

```
<!-- Declare a namespace -->
<td:elements
    xmlns:td="http://univ-littoral.fr/namespaces/m1/td">
  <elt attr1="..." attr2="..." />
  <elt attr1="..." attr2="..." />
</td:elements>
```

```
<!-- Multiple namespaces -->
<td:elements
    xmlns:td="http://univ-littoral.fr/namespaces/m1/td"
  xmlns:tp="http://univ-littoral.fr/namespaces/m1/tp">
  <tp:elt attr1="..." attr2="..." />
  <tp:elt attr1="..." attr2="..." />
</td:elements>
```

Espaces de noms

```
1 <!-- Default namespaces -->
2 <elements
      xmlns="http://univ-littoral.fr/namespaces/m1/td">
3    <elt attr1="..." attr2="..." />
4    <elt attr1="..." attr2="..." />
5 </elements>
```

Attention

Une fois un élément/attribut est associé à un espace de noms, son nom complet devient {URI}nom_element (Convention mais utilisée par les différents analyseurs/parsers)
.e.g le nom de l'élément "elt" ci-dessus associé à l'espace de noms par défaut est:
{http://univ-littoral.fr/namespaces/m1/td}elt

Conclusion

- Un moyen de regouper les éléments et les attributs sous un en-tête commun.
- On peut utilisé n'importe quel nom, mais c'est conseillé d'utiliser un URI.
- On peut choisir entre un espace de noms par défaut dont les éléments sont associés automatiquement ou une declaration d'espace de noms préfixés.
- Le préfixe peut utiliser n'importe quelle chaîne de caractères contenant pas **":"**. Pour une question de praticité, on utilise une chaine simple et courte.

Exercice

Exercice 3 - Espaces de noms

- Déclarer un espace de noms par défault dans l'element racine du document XML de l'exercice 1 (Recettes).
- Déclarer un espace de noms préfixé (prefix et url au choix), et associé certains éléments à l'espace déclaré.

DTD?

La rôle d'une DTD (Document Type Definition) est de *définir* la *structure d'un document*. Il s'agit d'un certain nombre de contraintes que doit respecter un document pour être valide. Ces contraintes spécifient:

- Les éléments qui peuvent apparaître dans le contenu d'un élément
- L'ordre éventuel de ces éléments.
- La présence de texte brut.
- Les attributs autorisés et les attributs obligatoires pour chaque éléments.

Pourquoi utiliser une DTD?

Example d'Utilisation

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Document Type Definitions (DTD)

Example

```
1  <!ELEMENT bibliography (book)+> ①
2  <!ELEMENT book (title, author, year, publisher, isbn,
       url?)> ②
3  <!ATTLIST book key NMTOKEN #REQUIRED> ③
4  <!ATTLIST book lang (fr | en) #REQUIRED> ④
5  <!ELEMENT title (#PCDATA)> ⑤
6  <!ELEMENT author (#PCDATA)>
7  <!ELEMENT year (#PCDATA)>
8  <!ELEMENT publisher (#PCDATA)>
9  <!ELEMENT isbn (#PCDATA)>
10 <!ELEMENT url (#PCDATA)>
```

① Déclaration de l'élément bibliography devant contenir une suite non vide d'éléments book.
② Déclaration de l'élément book devant contenir les éléments title, author, ..., isbn et url.
③④ Déclarations des attributs obligatoires key et lang de l'élément book.
⑤ Déclaration de l'élément title devant contenir uniquement du texte.

Déclaration Interne

Lorsque la DTD est incluse dans le document, sa déclaration prend la forme suivante: <!DOCTYPE root-element [declarations]>
Utilisation utile en phase de dévélopement.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE name [
3     <!ELEMENT name (first, middle, last)>
4     <!ELEMENT first (#PCDATA)>
5     <!ELEMENT middle (#PCDATA)>
6     <!ELEMENT last (#PCDATA)>
7  ]>
8  <name>
9     <first>Joseph</first>
10    <middle>John</middle>
11    <last>Fawcett</last>
12 </name>
```

Déclaration Externe

Lorsque la DTD est dans un fichier externe, on donne l'adresse du DTD.
Privée: <!DOCTYPE root-element SYSTEM "url" [declarations]>
Public: <!DOCTYPE root-element PUBLIC "fpi" "url" [declarations]>

Formal Public Identifier (fpi)

format: type//owner//desc//lang
-//W3C//DTD XHTML 1.0 Strict//EN
ISO//IEC 10179:1996//DTD DSSSL Architecture//EN

```
1 <?xml version="1.0"?>
2 <!DOCTYPE name SYSTEM "name.dtd" []>
3 <name>
4    <first>Joseph</first>
5    ...
6 </name>
```

Contenu de la DTD

Une DTD est généralement constitué de trois type de déclarations:

- Déclaration des Eléments. **ELEMENT**
- Déclaration des Attributs. **ATTLIST**
- Déclaration des Entités. **ENTITY**

Examples

- element <message>Bonjour</message>
- attribut
- entités: < > & " '

Déclaration des Elements

<!ELEMENT element−name category> ou
<!ELEMENT element−name (element−content)>

Example

Un seul occurrence : <!ELEMENT contact (name)>
plusieurs occurrence : <!ELEMENT contact (name+)>
Sequennce: <!ELEMENT name (first, middle, last)>
Choix: <!ELEMENT location (address | GPS)>
Combinaison: <!ELEMENT location (address | (latitude, longitude))>
Text: <!ELEMENT first (#PCDATA)>
Mixte: <!ELEMENT description (#PCDATA | title | detail)*>
Vide: <!ELEMENT br EMPTY>
Any: <!ELEMENT description ANY>

Cardinalité

+ Indique qu'un élément peut apparaître une ou plusieurs fois.

*: zero ou plus, ?: zero ou un, none: un

Déclaration des Attributs

<!ATTLIST element−name attribute−name attribute−type attribute−value>

Type (attribute-type)

- CDATA: text (default).
- ID: unique.
- IDREF: reference par ID.
- IDREFS: liste des valeurs d'IDREF séparés par des espaces.
- ENTITY: entité externe.
- ENTITIES: liste ENTITY.
- NMTOKEN: token name.
- NMTOKENS: liste NMTOKEN
- (en1—en2—..) Liste énumérée des valeurs possibles.

Déclaration des Attributs

<!ATTLIST element−name attribute−name attribute−type attribute−value>

Valeur (attribute-value)

- valeur par défault <!ATTLIST phone kind (Home | Work)"Home">
- valeur fixée <!ATTLIST contacts version CDATA #FIXED "1.0">
- requis <!ATTLIST phone kind (Home | Work)#REQUIRED>
- optionel <!ATTLIST knows contacts IDREFS #IMPLIED>

Entités

Les entités sont utilisés dans les documents XML pour faire référence à de simple charactères (Entités prédéfinies), des sections de texte, d'autre balises, et même des fichiers externes. On peut séparer les entités en quatre types principaux.

- Entités prédéfinies < (&), >(>), & (<), ' ('), &quot (")
  <description>Author & programmer</description>
- Entités de Caractères: &#unicode;
  <price>30 €</price>
- Entités Générales
  <!ENTITY name "fragment">
  <!ENTITY titre "Cours XML">
  <title>&titre;</title>
- Entités de Paramètres (Uniquement dans DTD):
  <!ENTITY % name "fragment">
  <!ENTITY % DefaultPhoneKind '"Home"'>
  <!ATTLIST phone kind (Home | Work)"%DefaultPhoneKind;">

Exercices

- Exo 4 - Créer un fichier DTD pour valider le document XML de l'exercice 1 (Recettes).
- Exo 5 - Créer un fichier DTD pour valider le document XML de l'exercice 2 (Fabrication du Champagne).

XML Schemas

XML Schemas?

Comme la DTD mais d'une facon plus précise, les XML Schemas permettent de *définir* la *structure d'un document*.

Avantages

- La syntaxe des schémas est une syntaxe purement XML.
- Schémas XML prennent en charge la recommandation d'espaces de noms.
- Schémas XML permettent de valider le contenu des éléments en fonction des types des données: prédéfinis (chaîne de caractère, entier, floattant, date, ...).
- Schémas XML permettent de définir nouveaux types des données (entiers entre 1 et 12).
- Les schémas XML vous permettent de créer plus facilement des modèles de contenu complexes et réutilisables.
- Les schémas XML permettent la modélisation de concepts de programmation tels que l'héritage d'objets.

XML Schemas

Example

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"> ①
3  <xsd:element name="bibliography" type="Bibliography"/> ②
4  <xsd:complexType name="Bibliography"> ③
5     <xsd:sequence> ④
6        <xsd:element name="book" minOccurs="1" maxOccurs="unbounded"> ⑤
7           <xsd:complexType>
8              <xsd:sequence>
9                 <xsd:element name="title" type="xsd:string"/>
10                <xsd:element name="author" type="xsd:string"/>
11                <xsd:element name="year" type="xsd:string"/>
12                <xsd:element name="publisher" type="xsd:string"/>
13                <xsd:element name="isbn" type="xsd:string"/>
14                <xsd:element name="url" type="xsd:string" minOccurs="0"/>
15             </xsd:sequence>
16             <xsd:attribute name="key" type="xsd:NMTOKEN" use="required"/> ⑥
17             <xsd:attribute name="lang" type="xsd:NMTOKEN" use="required"/> ⑦
18          </xsd:complexType>
19       </xsd:element>
20    </xsd:sequence>
21 </xsd:complexType>
22 </xsd:schema>
```

- ① Élément racine xsd:schema avec la déclaration de l'espace de noms des schémas associé au préfixe xsd.
- ② Déclaration de l'élément bibliography avec le type Bibliography.
- ③ Début de la définition du type Bibliography.
- ④ Définition du type comme une suite d'autres éléments (Ordre est important).
- ⑤ Déclaration de l'élément book dans le contenu du type Bibliography.
- ⑥⑦ Déclaration des attributs key et lang de l'élément book avec le type xsd:NMTOKEN.

XML Schemas

Structure Globale d'un Schéma

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Declarations d'elements, d'attributs et definitions de types -->

</xsd:schema>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<!-- Declarations d'elements, d'attributs et definitions de types -->

</schema>
```

Globaux ou Locaux, Nommés ou Anonymes

- globaux si leur définition est un enfant direct de xsd:schema
- sinon **Anonymes**

- Les éléments et les attributs sont toujours **nommés**
- Les types sont nommés s'ils sont globaux.

Il est possible dans un document de donner explicitement le schéma devant servir à le valider. On utilise un des attributs *schemaLocation* ou *noNamespaceSchemaLocation* dans l'**élément racine du document à valider**.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Reference vers un schEma servant A valider ce document xml utilisant des
       espaces des noms -->
3  <bibliography xsi:schemaLocation="http://example.org/~bibliography/
       bibliography.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4  <!-- notre format xml -->
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Reference vers un schEma servant A valider ce document xml sans utilisation
       des espaces de noms -->
3  <bibliography xsi:noNamespaceSchemaLocation="bibliography.xsd"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5  <!-- notre format xml -->
```

Types prédéfinis

- Numériques: *boolean*, *byte*, *unsignedByte*, *short*, *int*, *long*, *float*, *double*, *decimal*, ...
- Text: *string*, *normalizedString*, *token*, *Name*, *QName*, *NCName*, *language*, *anyURI*, *base64Binary*, *hexBinary*, ...
- Date et Heure: *time*, *date*, *dateTime*, *duration*, *dayTimeDuration*, *yearMonthDuration*, ...
- Hérité des DTD: *ID*, *IDREF*, *IDREFS*, *NMTOKEN*, *NMTOKENS*, *ENTITY*, *ENTITIES*, *NOTATION*

Déclarer un élément:

```
<element
  abstract = Boolean : false
  block = (#all | List of (extension | restriction | substitution))
  default = string
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified)
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = Boolean : false
  ref = QName
  substitutionGroup = QName
  type = QName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, ((simpleType | complexType)?, (unique | key |
keyref)*))
</element>

<!-- Parent Elements: schema, choice, all, sequence -->
```

Attributs

- absract (true — false): indique si l'élément peut être utilisé dans une instance d'un document.
- default: valeur par defaut
- substitutionGroup: utilisé quand abstract est vrai.
- fixed: valeur fixé
- form: qualified: l'élémnt doit etre associé à un espace de noms.
- id: id de type ID
- maxOccurs, minOccurs
- name
- nillable: si l'élément peut prendre la valeur nil
- ref: le nom d'un élément présent dans le schéma.
- type: type prédéfini ou nom d'un simpleType ou complexType présent dans le schema.

XML Schemas

Déclarations d'éléments: element

```
<xs:element name="cat" type="xs:string"/>
<xs:element name="dog" type="xs:string"/>
<xs:element name="redDog" type="xs:string"
    substitutionGroup="dog" />
<xs:element name="brownDog" type="xs:string"
    substitutionGroup ="dog" />

<xs:element name="pets">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="cat"/>
      <xs:element ref="dog"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

XML Schemas

Déclarations d'Elément: attribute

```
<!-- Pour declarer un attribut -->
<attribute
  default = string
  fixed = string
  form = (qualified | unqualified)
  id = ID
  name = NCName
  ref = QName
  type = QName
  use = (optional | prohibited | required): optional
  {any attributes with non-schema Namespace...}>
Content: (annotation?, (simpleType?))
</attribute>

<!-- Parent elements: attributeGroup, schema, complexType, restriction
     (simpleContent), extension (simpleContent), restriction (complexContent),
     extension (complexContent) -->

<xs:complexType name="myComplexType">
  <xs:attribute name="mybaseattribute" type="xs:string" use="required"/>
</xs:complexType>

<xs:attribute name="mybaseattribute" type="xs:string" default="test" />
<xs:complexType name="myComplexType">
  <xs:attribute ref="mybaseattribute"/>
</xs:complexType>
```

XML Schemas

Déclarations d'Elément: attribute

```
<!-- Pour declarer un attribut -->
<!-- Declarer un attribut ayant un valeur entre 60 et 95 -->
<xs:attribute name="myHolidayLocationTemperature">
  <xs:simpleType>
   <xs:restriction base="xs:integer">
    <xs:minInclusive value="60"/>
    <xs:maxInclusive value="95"/>
   </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<!-- Attribut comme liste de decimaux -->
<xs:simpleType name="Sizes">
   <xs:restriction base="xs:decimal">
      <xs:enumeration value="10.5"/>
         <xs:enumeration value="9"/>
      <xs:enumeration value="8"/>
      <xs:enumeration value="11"/>
   </xs:restriction>
</xs:simpleType>

<xs:attribute name="shoeSizes">
   <xs:simpleType>
      <xs:list itemType="Sizes"/>
   </xs:simpleType>
</xs:attribute>
```

XML Schemas

Déclarations d'éléments: annotation

```
<annotation
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (appinfo | documentation)*
</annotation>

<! -- Parent Elements: Any element -->

<!-- Remarque: Une annotation peut contenir des elements 'appinfo' (informations
     A utiliser par les applications) et des elements 'documentation'
     (commentaires ou texte A lire ou A utiliser par les utilisateurs). -->

<xs:simpleType name="northwestStates">
    <xs:annotation>
        <xs:documentation>States in the Pacific Northwest of US</xs:documentation>
    </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value='WA'>
      <xs:annotation>
        <xs:documentation>Washington</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    ...
  </xs:restriction>
</xs:simpleType>
```

XML Schemas

Déclarations d'éléments: simpleType

```
<simpleType
  id = ID
  name = NCName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (restriction (simpleType) | list | union))
</simpleType>

<! -- Parent Elements: attribute, element, list, restriction (simpleType),
      schema, union -->

<xs:simpleType name="listOfDates">
  <xs:list itemType="xs:date"/>
</xs:simpleType>

<xs:simpleType>
 <xs:union>
  <xs:simpleType>
   <xs:restriction base="roadbikesize"/>
  </xs:simpleType>
  <xs:simpleType>
   <xs:restriction base="mountainbikesize"/>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>
```

XML Schemas

Déclarations d'éléments: restriction

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (simpleType?, (minExclusive | minInclusive |
maxExclusive | maxInclusive | totalDigits |fractionDigits | length |
minLength | maxLength | enumeration | whiteSpace | pattern)*)?,
((attribute | attributeGroup)*, anyAttribute?))
</restriction>

<! -- Parent Elements: simpleType, simpleContent, complexContent -->

<xs:simpleType name="mountainbikesize">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="FamilyMountainBikeSizes">
   <xs:simpleContent>
      <xs:extension base="mountainbikesize">
         <xs:attribute name="familyMember" type="xs:string" />
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

XML Schemas

Déclarations d'éléments: extension

```
<extension
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, ((attribute | attributeGroup)*, anyAttribute?))
</extension>

<! -- Parent Elements: simpleContent, simpleType, complexContent -->

<!-- The following example extends a defined simpleType by adding an enumerated
     attribute. -->
<xs:complexType name="FamilyMountainBikes">
  <xs:simpleContent>
    <xs:extension base="mountainBikeSize">
      <xs:attribute name="familyMember">
      <xs:restriction base="xs:string">
        <xs:enumeration value="child" />
        <xs:enumeration value="male" />
        <xs:enumeration value="female" />
      </xs:restriction>
    </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

XML Schemas

Déclarations d'éléments: list

```
<list
  id = ID
  itemType = QName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (simpleType?))
</list>
<! -- Parent Elements: simpleType -->

<!-- The following example shows a simpleType that is a list of integers. -->
<xs:simpleType name='listOfIntegers'>
  <xs:list itemType='integer'/>
</xs:simpleType>
```

XML Schemas

Déclarations d'éléments: union

```
<union
  id = ID
  memberTypes = List of QNames
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (simpleType*))
</union>
<! -- Parent Elements: simpleType -->

<xs:attribute name="fontsize">
  <xs:simpleType>
    <xs:union memberTypes="fontbynumber fontbystringname" />
  </xs:simpleType>
</xs:attribute>

<xs:simpleType name="fontbynumber">
  <xs:restriction base="xs:positiveInteger">
    <xs:maxInclusive value="72"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="fontbystringname">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small"/>
    <xs:enumeration value="medium"/>
    <xs:enumeration value="large"/>
  </xs:restriction>
</xs:simpleType>
```

XML Schemas

Déclarations d'éléments : complexType

```
<complexType
  abstract = Boolean : false //
  block = (#all | List of (extension | restriction))
  final = (#all | List of (extension | restriction))
  id = ID
  mixed = Boolean : false
  name = NCName
  {any attributes with non-schema Namespace...}>
Content: (annotation?, (simpleContent | complexContent | ((group | all |
choice | sequence)?, ((attribute | attributeGroup)*, anyAttribute?))))
</complexType>
<! -- Parent Elements: element, redefine, schema -->

<xs:element name='myShoeSize'>
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base='xs:decimal'>
     <xs:attribute name='sizing' type='xs:string' />
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

XML Schemas

Déclarations d'éléments: sequence

```
<sequence
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (element | group | choice | sequence | any)*)
</sequence>

<! -- Parent Elements: group, choice, sequence, complexType, restriction
     (simpleContent), extension (simpleContent), restriction (complexContent),
     extension (complexContent) -->

<!-- The following example shows an element (zooAnimals) that can have zero or
     more of the following elements, elephant, bear, giraffe, in the sequence
     element. -->
<xs:element name="zooAnimals">
     <xs:complexType>
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
             <xs:element name="elephant"/>
             <xs:element name="bear"/>
             <xs:element name="giraffe"/>
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

XML Schemas

Déclarations d'éléments: choice

```
<choice
  id = ID
  maxOccurs= (nonNegativeInteger | unbounded) : 1
  minOccurs= nonNegativeInteger : 1
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (element | group | choice | sequence | any)*)
</choice>
<! -- Parent Elements: group, choice, sequence, complexType, restriction
     (simpleContent), extension (simpleContent), restriction (complexContent),
     extension (complexContent) -->

<xs:complexType name="chadState">
 <xs:choice minOccurs="1" maxOccurs="1">
  <xs:element ref="selected"/>
  <xs:element ref="unselected"/>
  <xs:element ref="dimpled"/>
  <xs:element ref="perforated"/>
 </xs:choice>
 <xs:attribute name="candidate" type="candidateType"/>
</xs:complexType>
```

XML Schemas

Déclarations d'éléments : group

```
<all id = ID
  maxOccurs= 1: 1 minOccurs= (0 | 1): 1
  {any attributes with non-schema Namespace...}>
Content: (annotation?, element*)
</all>
<! -- Parent Elements: group, restriction, extension, complexType -->

<?xml version="1.0"?>
<myElement myAttribute="1.1">
    <thing2>Some</thing2>
    <thing3>text</thing3>
    <thing1>for you</thing1>
</myElement>

<xs:element name="thing1" type="xs:string"/>
<xs:element name="thing2" type="xs:string"/>
<xs:element name="thing3" type="xs:string"/>
<xs:attribute name="myAttribute" type="xs:decimal"/>
<xs:complexType name="myComplexType">
 <xs:all>
  <xs:element ref="thing1"/>
  <xs:element ref="thing2"/>
  <xs:element ref="thing3"/>
 </xs:all>
 <xs:attribute ref="myAttribute"/>
</xs:complexType>
```

XML Schemas

Déclarations d'éléments

```
<group
  name= NCName
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  ref = QName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (all | choice | sequence))
</group>
<!-- Parent elements: schema, choice, sequence, complexType, restriction
     (complexContent), extension (complexContent) -->

<!-- Definir un groupe et l utiliser dans une definition d un type complexe.-->
<xs:attribute name="myAttribute" type="xs:decimal"/>

<xs:group name="myGroupOfThings">
 <xs:sequence>
   <xs:element name="thing1" type="xs:string"/>
   <xs:element name="thing2" type="xs:string"/>
 </xs:sequence>
</xs:group>

<xs:complexType name="myComplexType">
  <xs:group ref="myGroupOfThings"/>
  <xs:attribute ref="myAttribute"/>
</xs:complexType>
```

Example utilisant les esapces de Noms:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:target="http://www.example.com/name"
   targetNamespace="http://www.example.com/name"
   elementFormDefault="qualified">
   <complexType name="NameType">
      <sequence>
         <element name="first" type="string"/>
         <element name="middle" type="string"/>
         <element name="last" type="string"/>
      </sequence>
      <attribute name="title" type="string"/>
   </complexType>
   <element name="name" type="target:NameType"/>
</schema>

<name xmlns="http://www.example.com/name"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.example.com/name name.xsd">
   <first>John</first>
   <middle>Fitzgerald</middle>
   <last>Doe</last>
</name>
```

Attributs de l'élément xsd:schema

DOM & XPath

XSLT

XQuery

XML and Databases

Sax

XmlReader

LINQ to XML

RSS, ATOM

SOAP

AJAX

XHTML

Scalable Vector Graphics (SVG)

AJAX