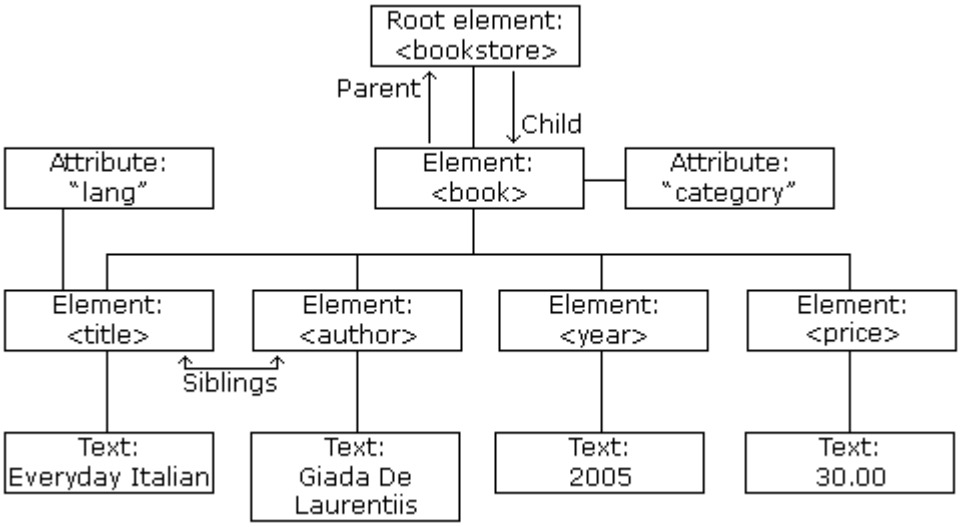
Résumé XML.

XML(Extensible Markup Language / extension.xml)

- eXtensible Markup Languae.
- Créer pour standardiser l'échange et le stockage des données (et non pour l'affichage).
- Arborescence des données.

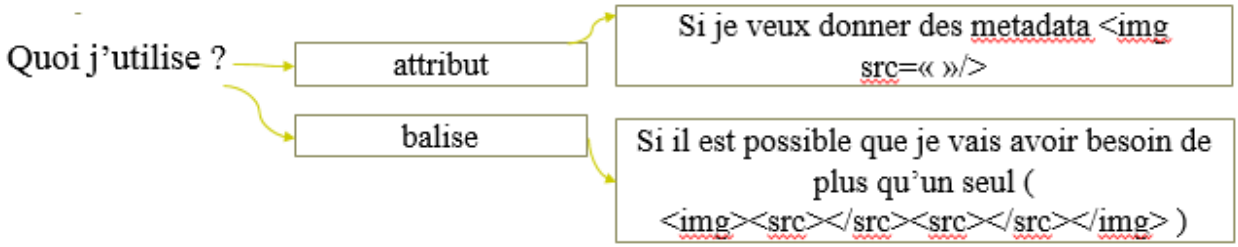


- Un document XML doit avoir un nœud RACINE
- Les balises XML doivent être correctement imbriquées
- Les balises XML doivent être fermées
- Les attributs de balises XML doivent être entourés de guillemets
- Les balises XML sont sensibles à la casse.
- Un document XML qui respecte ces règles est dit bien formé. (Well Formed).

&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	"

- < !—Comment -->
- Les attributs sont utilisés pour donner des informations sur les données.
- Les attribues sont aussi des child (fils) d'un parent, la principale différence entre les attributs et les balises, est que l'attribut ne peut pas contenir des enfants.

```
<img src=""></img>
<img>
   <src></src>
   <src></src>
</img>
```



- Déclaration XML importante

```
<?xml version="1.0" encoding="utf-8"  standalone="yes"?>
```

- **version** : version du XML 1.0
- **encoding** : Codage du jeu de caractères utilisé
- **standalone** : autonomie du document par rapport à une **dtd** externe (par défaut "no")
- Le fichier doit commencer par cette déclaration.
- Add css to XML

```
<?xml-stylesheet type="text/css" href="news.css"?>
```

DTD (Document Type Definition / extension.dtd)

- DTD: Document Type Definition
- Définie les éléments autorisés dans un fichier XML (éléments, attributs, entité).
- Objectifs :
  Standardiser l'écriture du fichier.
  Valider la structure des données avant les consommer.
- Un DTD peut être interne (définie dans le fichier XML) ou externe (définie à l'extérieur et appelé u fichier XML).
  // **syntaxe**
  <!DOCTYPE élement-racine [déclaration-d-éléments]>
  // **exemple**
  <!DOCTYPE repertoire [

<!ELEMENT repertoire (contact)> élément répertoire doit contenir l'élément contact
<!ELEMENT contact (nom, prenom?, tele+, adresse-mail*)>
l'élément contact doit contenir : nom obligatoirement une seule fois, prénom facultatif soit une fois soit sans prénom,, télé une fois au minimum, adresse-mail plusieurs fois ou sans adresse-mail
<!ELEMENT nom (#PCDATA)>
PC DATA = parsed character data (à traduire si il est une entité...)
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT tele (#PCDATA)>
<!ELEMENT adresse-mail (#PCDATA)>
]>

- Déclaration d'un dtd externe.
<!-- <!DOCTYPE element-racine SYSTEM "nom-fichier"> -->
<!DOCTYPE repertoire SYSTEM "repertoire.dtd" >
- Identifier un élément et ses élément enfant :

<!ELEMENT nom-element categorie> ou <!ELEMENT nom-element (contenu-element)>

<!ELEMENT contact (nom, prenom?, tele+, adresse-mail*)>

Catégorie	Signification
EMPTY	Elément vide
ANY	Elément avec n'importe quel contenu

- Contenue

Contenue	Signification
(#PCDATA)	Elément avec contenu paré (contenu à traduire).
(enfant1)	Elément doit contenir l'élément <enfant1> une seule fois.
(enf1, enf2)	Elément doit contenir <enf1> et <enf2> dans cet ordre, une seule fois.
(enfant1 ?)	Elément peut contenir l'élément <enfant1> **0 ou 1** (une) seule fois.
(enf1|enf2)	Elément doit contenir soit <enf1> soit <enf2>.
(enfant1 +)	Elément peut contenir l'élément <enfant1> une seule fois au minimum (**1 ou plusieurs**).
(enfant1 *)	Elément peut contenir l'élément <enfant1> 0 fois ou plusieurs (**0 ou plusieurs**).
(enf1|enf2)*	== **(enf1*|enf2*)**
(#PCDATA, enf1*, enf2 ?)	Contenue mixte, **#PCDATA** doit être au début.

- "#PCDATA means that the element may contain parsed character data, which is text that the document processor actually looks at and interprets to display both content and markup".
- ***Déclarer un attribut.***

<!ATTLIST nom-element
nom-attribut
type-attribut
valeur-par-defaut
>

- On peut utiliser un type ou bien une énumération des valeurs comme suite.

<!ATTLIST contact type (pro| perso) "pro">

Type	Description
CDATA	Chaine de caractère
(en1|en2|en3|…)	Une parmi les valeurs de la liste d'énumération
ID	Identificateur unique
IDREF	Référence à l'identificateur d'un autre élément
IDREFS	Liste de IDREF
NMTOKEN	Nom XML valide
NMTOKENS	Liste de NMTOKEN
ENTITY	Sa valeur est une entité
ENTITIES	Liste d'entités
NOTATION	La valeur de l'attribut est une NOTATION

- **CDATA** = Character data.

CDATA, or character data, enables the author to include any string of characters that doesn't include the ampersand (&), less-than and greater-than signs (< or >), or quotation marks (“). These four characters may be represented using character entities (&, <, >, or ", respectively).

• **ID** creates a unique ID for an attribute that identifies an element. This type is most often used by programs that process a document.

• **IDREF** allows the value of an attribute to be the same as the ID of an element somewhere else in the document.

• **IDREFS** is just like IDREF, but the value may be made up of multiple IDREFs.

• **ENTITY** allows you to use external binary data or unparsed entities. You'll get the scoop on entities in the next section of this chapter.

• **ENTITIES** allows you to link multiple entities.

• **NMTOKEN** restricts the value of the attribute to any valid XML name.

• **NMTOKENS** allows the value of the attribute to be composed of multiple XML names.

• **NOTATION** allows you to use a value already specified with a notation declaration in the DTD.

- **ID : digits (and some other characters) are disallowed as initial characters.**
- IDREFS usage

```
<auteur id="AZ123" id_publications="ad123 ac123">
        <nom>Essadeq ELAAMIRI</nom>
</auteur>
```

Type de valeur par defaut	Description
"valeur"	La valeur par défaut de l'attribut
#REQUIRED	Valeur requise
#IMPLIED	Valeur facultative
#FIXED "valeur"	Valeur fixée

- Les entités sont utilisées pour définir des raccourcis à du texte standard ou à des caractères spéciaux.
- Déclarer entités.

// interne (from xml file or dtd file associated).

```
<!--  <!ENTITY nom "valeur"> -->
```

```
<!ENTITY gml "@gmail.com" >
```

// externe (from an other dtd file)

```
<!ENTITY ent SYSTEM "file.dtd">
```

Schémas XML (XML Schema Definition / extension.xsd)

- Les limites de DTD :
  - **Les DTD ne sont pas de type XML** : Il 'est nécessaire d'utiliser un outil spécial pour "parser" un tel fichier, différent de celui utilisé pour le "parsing" des fichiersXML.
  - **Le DTD ne supporte pas les « espaces de nom »** : Cela implique qu'il n'est pas possible, dans un fichier XML défini par une DTD, d'importer des définitions de balises définies par ailleurs.
  - **Le "typage" des données est limité**.
- En plus des fonctionnalités fournies par les DTD, des nouveautés :
  - Le typage des données (types prédéfinie ou bien nouveaux types).
  - La notion de l'héritage.
  - Le support des espaces de nom.
  - Les fichiers de schéma XML sont des fichiers XML : qu'un parseur XML permet de les manipuler facilement.
- **Lier un fichier XSD au fichier XML :**

Il faut ajouter les deux attributs suivants au élément racine du fichier XML.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="library.xsd"
```

Le premier indique que le fichier XML est une instance du fichier XSD. Et le deuxième pour localiser le fichier schéma.

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

- **Structure :**
  Un prologue : <?xml version="1.0" encoding="UTF-8"?>
  Elément racine : <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"></xs:schema>
- L'élément racine est : <xs:schema>
- Les éléments du XML schéma sont définis dans l'espace nom : http://www.w3.org/2001/XMLSchema, qu'est préfixé ici par «xs ».

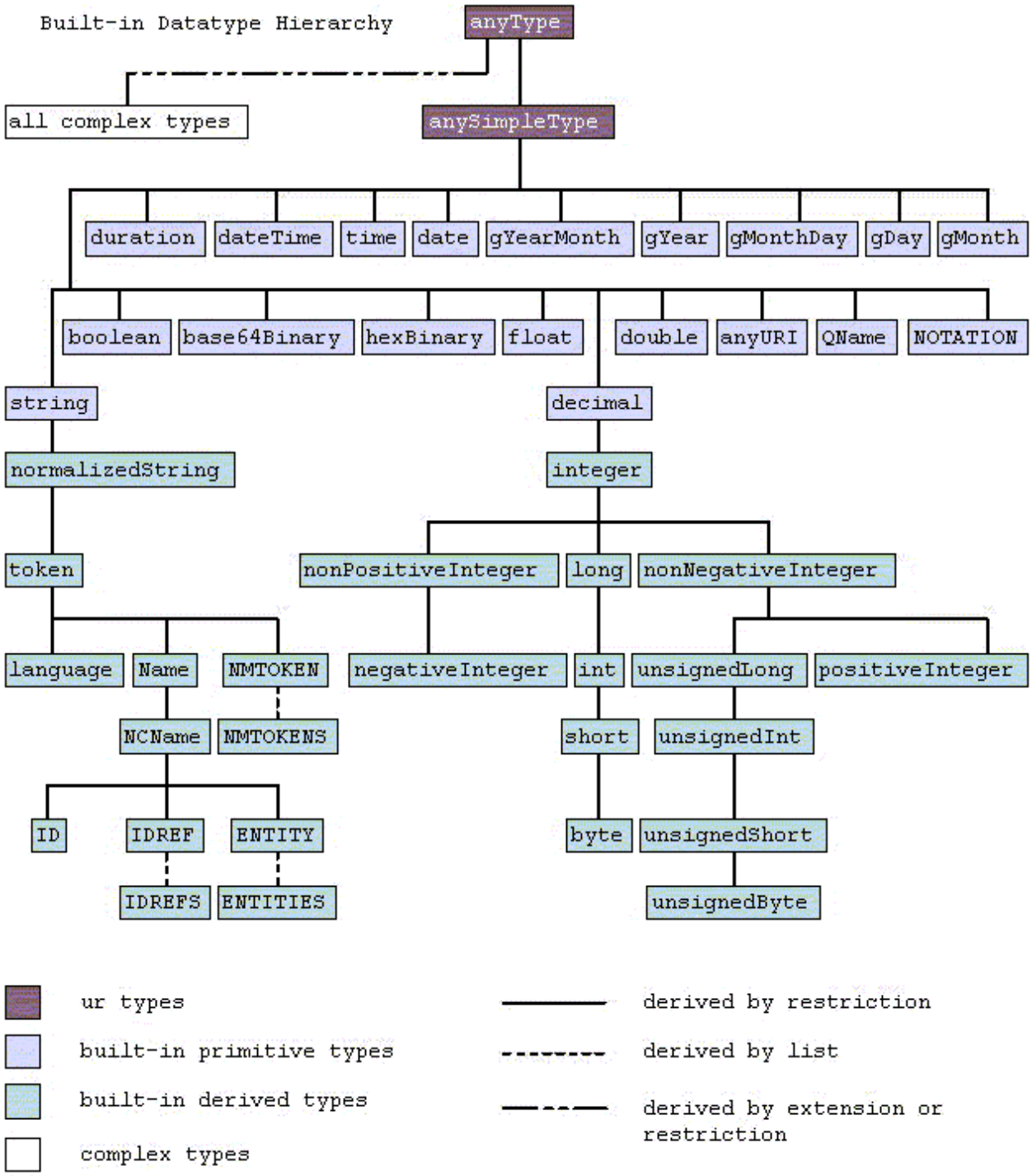
By using the format xmlns:xsd**, you indicate that any elements or attributes with an** xsd: **prefix belong to this namespace (**http://www.w3.org/2001/XMLSchema**).**

In fact, if you're only using one namespace, you don't have to use a prefix at all! Prefixes are used to distinguish between two or more namespaces.

- Cela signifie que tous les éléments de XML schéma commencent par xs (<xs :element>).
- **Déclaration d'un élément :**

<xs:element name="nom" type="typeElem" [default="" fixed=""]></xs:element>

- Un élément peut être de **type simple** (contient que des TextNodes [les types prédéfinis, ou des types personnalisés par des restrictions, listes…], il ne peut contenir ni autres éléments XML ni attributs), ou bien de **type complexe** (dans le cas où l'élément peut contenir des autres éléments et des attributs).
- **Elément simple** : <xs:element name="price" type="xs:float" ></xs:element>
- Voici les types de données supportés par XML schemas.



- **<u>Les restrictions dans un élément simple :</u>**

Un élément simple peut contenir des restrictions.

Par exemple je veux que la valeur de l'élément 'price' soit compris entre 10$ et 199$.

```
<xs:element name="price" >
          <xs:simpleType>
```

```
                <xs:restriction base="xs:float">
                    <xs:minInclusive value="10"></xs:minInclusive>
                    <xs:maxExclusive value="200"></xs:maxExclusive>
                 </xs:restriction>
            </xs:simpleType>
    </xs:element>
```

- Les restrictions permettent de la **dérivation** des nouveaux types simples à partir les types simple existants (la base), en appliquant des « facettes », qui sont des contraintes supplémentaires appliquées un type simple particulier.
- Une "facette" permet de placer une contrainte sur l'ensemble des valeurs que peut prendre un type de base.
- **Restrictions on Values :**

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Restrictions on a Set of Values:**

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Restrictions on a Series of Values:**

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint (using regular expressions).

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
```

```
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Restrictions on Whitespace Characters :**

To specify how whitespace characters should be handled, we would use the whiteSpace constraint.

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The value it can be preserve (which means that the XML processor WILL NOT remove any white space characters), replace (which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces), or collapse (which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space)).

- **Restrictions on Length :**

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Or

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
```

```
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero

pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

- **Listes**

Il est possible de créer une liste personnalisée, par "dérivation" de types existants. Par exemple,

```
<!-- create a list of floats -->
    <xs:simpleType name="width_heigh">
        <xs:list itemType="xs:float">
        </xs:list>
    </xs:simpleType>
```

Using it in xml as : `<width_height>133.2 13.7 144.9</width_height>`

- **Unions**

On peut désirer, par exemple, qu'un type autorise soit un nombre, soit une chaîne de caractères particuliers.

Il est possible de le faire à l'aide d'une déclaration d'union.

```
<xs:simpleType name="date_str">
        <xs:union memberTypes="xs:string xs:date"></xs:union>
    </xs:simpleType>
```

Les éléments suivants sont alors des "instances" valides de cette déclaration :

```
<date_str>18</ date_str >
< date_str >Pompiers</ date_str >
```

- **Eléments complexes :**

On peut les définir soit localement ou globalement (pour la réutilisation).

Un élément complexe peut etre :

- **Empty** : vide avec des attributs :

```
<xs:element name="br">
            <xs:complexType>
              <xs:attribute name="color" default="#000000"></xs:attribute>
```

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

- **Elements Only** : contient seulement des autres éléments.
- **Mixed** : contient des TextNodes au plus des elements :

On peut déclarer un élément **mixe**, qui peut contenir des TextNodes en plus des autres éléments :

```
<xs:element name="published">
      <xs:complexType mixed="true">

            ……
```

- **Text Only** : contient que de TextNodes avec des attributs :

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>

      < !–Extension/ restriction-->
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- **Déclaration d'un élément complexe avec un attribut(localement) :**

```
<xs:element name="library ">
  <!-- declaration des types -->
  <xs:complexType >
    <!-- declaration de contenu -->
    <xs:attribute name="id" type="xs:ID"></xs:attribute>
  </xs:complexType>

</xs:element>
```

- **Déclaration d'un élément complexe avec un attribut(globalement) :**

```
<xs:element name="library" type="Lib"></xs:element>
  <!-- declaration des types -->
  <xs:complexType name="Lib">
    <!-- declaration de contenu -->
    <xs:attribute name="id" type="xs:ID"></xs:attribute>
  </xs:complexType>
```

On a ici un avantage, on peut réutiliser le type « LIB » au cas de besoin.

- Ici on a déclaré un élément qui contient un attribut. L'élément donc est de type complexe c'est pour quoi on a déclaré le modèle de contenu de type à l'intérieure du <xs:complexType name="Lib">.
- **Remarque** : on peut baser un élément complexe sur un autre élément complexe existant et ajouter des éléments (notion d'extension).

- **Attributs :**

<xs:attribute name="nom" type="type" [use="" default="" fixed=""]></xs:attribute>

- Un attribut ne peut être que de type simple.
- Les attributs d'un élément attribut des schéma XML :

Attribut	
Use	indique la présence , il peut prendre pour valeur **required**(obligatoire), **optional**(facultatif) ou **prohibited** (ne doit pas apparaitre)
Default	pour indiquer la valeur par défaut
Fixed	indique que l'attribut est renseigné, la seule valeur que peut prendre l'attribut déclaré est celle de l'attribut fixed
<xs:attribute name="id" type="xs:ID" use="required"></xs:attribute> <xs:attribute name="date" type="xs:date" use="optional" default="2021-11-19"></xs:attribute>	

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"></xs:element>
              <xs:element name="author" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="name" type="xs:string"></xs:element>
                    <xs:element name="contact" type="xs:string"></xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:ID" use="required"></xs:attribute>
                </xs:complexType>
              </xs:element>
              <xs:element name="published" type="xs:date"></xs:element>
              <xs:element name="description" type="xs:string"></xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID"></xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- declaration des types -->
</xs:schema>
```

- L'exemple ci-dessus on a créé un élément library qui contient des éléments book.
- Dans les types complexes on peut utiliser :
- **Indicateurs d'ordre :**

On trouve :

- **Sequence** : définie une suites d'éléments ordonnée (comme (elm1, elm2, em3) du DTD).

Dans cet exemple, tous les éléments de la séquence doivent être présents une et une seule fois et dans cet ordre.

```
<xs:element name="author" maxOccurs="unbounded">
    <xs:complexType>
       <xs:sequence>
          <xs:element name="name" type="xs:string"></xs:element>
          <xs:element name="contact" type="xs:string"></xs:element>
       </xs:sequence>
       <xs:attribute name="id" type="xs:ID" use="required">

        </xs:attribute>
     </xs:complexType>
</xs:element>
```

- **Choice** : énumère des choix (l'équivalent de (elm1|elm2|elm3) de DTD).

On peut ici utiliser un de ces éléments une seule fois.

```
<xs:element name="published">
  <xs:complexType>
   <xs:choice>
     <xs:element name="dateTime" type="xs:dateTime"></xs:element>
     <xs:element name="date" type="xs:date"></xs:element>
    </xs:choice>
   </xs:complexType>
 </xs:element>
```

- **All** : indique que les éléments enfants doivent apparaître une fois (~~ou pas du tout~~) dans n'importe quel ordre.

Cet élément *xsd:all* doit être un enfant direct de l'élément *xsd:complexType*.

```
<xs:element name="dateTime">
    <xs:complexType>
       <xs:all>
          <xs:element name="date" type="xs:date"></xs:element>
         <xs:element name="time" type="xs:time"></xs:element>
       </xs:all>
     </xs:complexType>
   </xs:element>
```

- **Indicateurs d'occurrence :**

On peut indiquer le nombre minimum et le nombre maximum de fois qu'il doit apparaître Indicateurs de groupes :

On utilise pour cela **minOccurs** (par défaut = 1), et **maxOccurs**(par défaut=1), on peut utiliser des valeurs positives ou bien (pour **maxOccurs**) '**unbounded'**== infini.

L'exemple suivant équivalant à (* de DTD).

```
<xs:sequence>
        <xs:element name="book" minOccurs="0" maxOccurs="unbounded"></xs:element>
</xs:sequence>
```

- ## Indicateurs de groupes :

On peut nommer des groupes d'éléments ou des attributs pour pouvoir les réutiliser.

- **Group** : il sert à factoriser les parties communes à plusieurs types, on doit nécessairement utiliser <sequence>, <choice> ou <all> à l'interieure.

```
<xs:group name="person_name_contact">
    <xs:sequence>
      <xs:element name="name" type="xs:string"></xs:element>
      <xs:element name="contact" type="xs:string"></xs:element>
    </xs:sequence>
  </xs:group>
```

On peut donc réutiliser ce groupe avec n'importe qu'il élément.

```
<xs:element name="author" maxOccurs="unbounded">
  <xs:complexType>
    <xs:group ref="person_name_contact"></xs:group>
    <xs:attribute name="id" type="xs:ID" use="required"></xs:attribute>
  </xs:complexType>
</xs:element>
```

- **AttributGroup** :

```
<xs:attributeGroup name="id_group">
    <xs:attribute name="id" type="xs:ID" use="required"></xs:attribute>
  </xs:attributeGroup>
```

Et pour l'utiliser :

```
<xs:attributeGroup ref="id_group"></xs:attributeGroup>
```

- ## Réutilisation d'un élément :

On peut aussi déclarer des éléments et les référencier dans plusieurs emplacements.

On déclare l'élément en dehors du root : <xs:element name="date" type="xs:date"></xs:element>

Et on l'utilise comme ceci : <xs:element ref="date"></xs:element>

- Extension d'un élément complexe :

On peut créer un type complexe en se basant sur un type existant et en ajoutant des éléments.

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XPATH (XML Path Language)

- Permet de sélectionner une partie d'un document XML
- L'exécution d'une expression Xpath retourne souvent une collection de nœuds.
- XPath uses "path like" syntax to identify and navigate nodes in an XML document
- XPath contains over 200 built-in functions.
- **Un chemin de localisation** est une expression XPath utilisée pour sélectionner une collection de nœuds par rapport au nœud de contexte.
- Le nœud de contexte est le nœud en cours d'évaluation.
- Un chemin de localisation se compose d'une ou de plusieurs étapes de localisation, séparées par des barres obliques (/).

locationstep/locationstep => chemin relatif

/locationstep/locationstep => chemin Absolu commence à partir de la racine.

- Un chemin de localisation peut être abrégé ou non. مختصر

- Dans un chemin de localisation non abrégé, les étapes de localisation adoptent la syntaxe suivante :

axis::node-test[predicate]

- **Axis(facultatif)** spécifie la relation entre les nœuds sélectionnés par l'étape de localisation et le nœud de contexte (la direction générale de l'étape).
- **node-test** spécifie le type de nœud ou le nom développé des nœuds sélectionnés par l'étape de localisation.
- **Predicate (facultatif)** est une expression de filtre qui affine la sélection des nœuds dans l'étape de localisation (un filtre spécifiant un critère de sélection).
  Child ::para[last()] => Sélectionne le dernier élément <para> du nœud de contexte.
  Parent ::para => Sélectionne l'élément <para> qui est parent du nœud de contexte.
  Child ::text()=> Sélectionne tous les nœuds de texte enfants du nœud de contexte.
  Child ::div/child ::para=> Sélectionne les éléments <para> enfants de l'élément qui est un enfant du nœud de contexte.
- Dans un chemin de localisation abrégé, l'axe, axis:: , n'est pas exprimé explicitement dans les étapes de localisation, mais implicitement par une série de raccourcis.
  ./div/para => Sélectionne les éléments <para> enfants de l'élément qui est un enfant du nœud de contexte.
  ../para => Sélectionne l'élément <para> qui est parent du nœud de contexte.
- Voici une synthèse de certaines abréviations :

Non abrégé	Abrégé
child ::*	*****
attribute ::*	**@***
/descendante-or-self ::node()	**//**
self ::node()	**.**
parent ::node()	**..**

- Le prédicat [position()=X] peut aussi abrévié par [X].
- Une étape de localisation sélectionne une collection de nœuds (node-set) par rapport au nœud de contexte.
- Exemples d'étape de localisation utilisant la syntaxe complète :

child::*[position()=1]	Localise le premier nœud enfant du nœud de contexte. >> ***context/*/[1]***
ancestor-or-self::book[@catdate="2000-12-31"]	Localise tous les ancêtres d'un enfant <book> du nœud de contexte, ainsi que l'enfant <book> lui-même, pour autant que l'élément en question possède un attribut catdate de valeur " 2000-12- 31 "
//parent::node()[name()="book"] | descendant::node()[name()="author"]	Localise n'importe quel nœud du document dont le nœud parent s'appelle « book » ou n'importe quel nœud descendant du nœud de contexte dont le nom est « author ».

- **<u>Les Axes :</u>**

Axes	**Description** (vers arrière / vers avant)
ancestor::	Ancêtres du nœud de contexte. أسلاف Les ancêtres du nœud de contexte sont constitués du parent du nœud de contexte et du parent du parent, etc. Donc, l'axe ancestor:: inclut toujours le nœud racine sauf si le nœud de contexte est le nœud racine.
ancestor-or-self::	Nœud de contexte et ses ancêtres. L'axe ancestor-or-self:: inclut toujours le nœud racine.
attribute::	Attributs du nœud de contexte. Cet axe est vide sauf si le nœud de contexte est un élément.
child::	Enfants du nœud de contexte. Un enfant est un nœud situé immédiatement sous le nœud de contexte dans l'arborescence. Toutefois, ni les nœuds d'attribut ni ceux d'espace de noms ne sont considérés comme des enfants du nœud de contexte.
descendant::	Descendants du nœud de contexte. Un descendant est un enfant ou un enfant d'un enfant, etc. Donc, l'axe descendant:: n'inclut jamais de nœud d'attribut ou d'espace de noms.
descendant-or-self::	Nœud de contexte et ses descendants.
following::	Tous les nœuds qui suivent le nœud de contexte dans l'arborescence, sauf les nœuds descendants, d'attribut et d'espace de noms.
following-sibling::	Tous les frères suivants du nœud de contexte. L'axe following-sibling:: identifie uniquement les enfants d'un nœud parent qui apparaissent dans l'arborescence après le nœud de contexte. Cet axe ne comprend pas les autres enfants qui apparaissent avant le nœud de contexte. Si le nœud de contexte est un nœud d'attribut ou d'espace de noms, l'axe following-sibling:: est vide.
namespace::	Nœuds d'espace de noms du nœud de contexte. Il existe un nœud d'espace de noms pour chaque espace de noms dans la portée du nœud de contexte. Cet axe est vide sauf si le nœud de contexte est un élément.
parent::	Le parent du nœud de contexte, le cas échéant. Le parent est le nœud situé juste avant le nœud de contexte dans l'arborescence.
preceding::	Tous les nœuds qui précèdent le nœud de contexte dans l'arborescence, sauf les nœuds ancêtres, d'attribut et d'espace de noms. L'une des manières de voir l'axe précédent est de considérer tous les nœuds dont le contenu se produit dans sa totalité avant le début du nœud de contexte.
preceding-sibling::	Tous les frères précédents du nœud de contexte. L'axe preceding-sibling:: identifie uniquement les enfants d'un nœud parent qui apparaissent dans l'arborescence avant le nœud de contexte. Cet axe ne comprend pas les autres enfants qui apparaissent après le nœud de contexte. Si le nœud de contexte est un nœud d'attribut ou d'espace de noms, l'axe preceding-sibling:: est vide.
self::	Nœud de contexte uniquement.

- **Les test_nodes :**

Nom du nœuds	Oriente la recherche vers le nom d'élément donné explicitement.
*****	Oriente la recherche vers tous les éléments du contexte défini par l'axe.
Node()	Oriente la recherche vers tous les types de nœuds (Eléments, Textes, Attributs, Commentaires etc…).
Text()	Oriente la recherche vers tous les types de nœuds Textes.
Comment()	Oriente la recherche vers tous les types de nœuds de Commentaires.

- **Fonctions XPath applicables aux nœuds (prédicats) :**

count()	Permet de compter le nombre de nœuds référencés. Par exemple, si dans un fichier XML on déclare la liste des 9 planètes du système solaire sous la forme de balises planete , l'instruction count(//planete) renverra la valeur 9.
position()	retourne la position du nœud contextuel.
last()	retourne le dernier nœud d'un ensemble de nœuds
name()	La fonction retourne le nom de l'élément,

- **Fonctions XPath applicables aux chaînes de caractères (prédicats):**

concat()	Concatène toutes les chaînes qui lui sont passées en arguments et retourne la chaîne résultant de cette concaténation
starts-with(chaine1, chaine2)	Renvoie la valeur true si chaine1 commence par la chaine2, false sinon
contains(chaine1, chaine2)	Renvoie true si chaine1 contient chaine2, false sinon.
substring(chaine1, decalage, longueur)	Retourne une sous-chaîne de chaine1 contenant longueur caractères et commençant à decalage.
substring-after(chaine1,chaine2)	Retourne la sous-chaîne de chaine1 qui suit la première occurence de chaine2.
substring-before(chaine1,chaine2)	Retourne la sous-chaîne de chaine1 qui précède la première occurrence de chaine2.
normalize-space()	Retourne la chaîne de caractères qu'elle reçoit en argument après En avoir supprimé les espaces situés au début et à la fin, et y avoir remplacé chaque séquence d'espaces successifs par un espace unique.
string-length()	Retourne la longueur de la chaîne qu'elle reçoit en argument.

- **Autres :**

+ , -, *	L'addition, la soustraction, et la multiplication.
div	Division
mod	Retourne le reste de la division euclidienne du premier nombre par le second.
ceiling(nombre)	Retourne le plus petit entier égal ou supérieur au nombre Qu'elle reçoit en argument ;
floor(nombre)	Retourne le plus grand entier égal ou inférieur au nombre Qu'elle reçoit en argument

round(nombre)	Arrondit le nombre qu'elle reçoit à l'entier le plus proche
sum(nombre1, nombre2, ...)	Retourne la somme des nombres reçus en argument. Cette fonction sert aussi à calculer la somme des éléments spécifiés par une expression xpath : par exemple, sum(//prix) calcule la somme des contenus de tous les éléments prix du document xml.
true() et false()	Créer une constante booléenne initialisée à une valeur true ou false
not(variable)	Inverse le sens logique de son argument
lang(chaine)	Vérifie que la langue dans laquelle est écrit le nœud courant (tel qu'elle est définie par l'attribut xml:lang) est le même que le langage qu'elle reçoit en argument. Cette fonction reçoit une chaîne correspondant à l'un des codes de langage définis dans la spécification xml : en pour l'anglais, jp pour le japonais, fr pour le français, etc
Number()	Convertir l'argument à un nombre.

IMAGES_EXAMPLES

Operator	Description
and	Logical-and
or	Logical-or
not()	Negation
=	Equality
!=	Not equal
< *	Less than
<= *	Less than or equal
> *	Greater than
<= *	Greater than or equal
|	Set operation; returns the union of two sets of nodes

* Extended XPath method

Expression	Refers to
author[last-name = "Bob"]	All <author> elements that contain at least one <last-name> element with the value Bob.
author[last-name[1] = "Bob"]	All <author> elements whose first <last-name> child element has the value Bob.
author/degree[@from != "Harvard"]	All <author> elements that contain <degree> elements with a from attribute that is not equal to "Harvard".
author[last-name = /editor/last-name]	All <author> elements that contain a <last-name> element that is the same as the <last-name> element inside the <editor> element under the root element.
author[. = "Matthew Bob"]	All <author> elements whose string value is Matthew Bob.

The |, or union, operator returns the union of its two operands, which must be node-sets. For example, //author | //publisher returns a node-set that combines all the //author nodes and all the //publisher nodes. Multiple union operators can be chained together to combine multiple node-sets. For example, //author | //publisher | //editor | //book-seller returns a node-set containing all //author, //publisher, //editor, and //book-seller elements. The union operator preserves document order and does not return duplicates.

Expression	Refers to
first-name | last-name	A node set containing <first-name> and <last-name> elements in the current context.
(bookstore/book | bookstore/magazine)	A node set containing <book> or <magazine> elements inside a <bookstore> element.
book | book/author	A node set containing all <book> elements and all <author> elements within <book> elements.
(book | magazine)/price	The node set containing all <price> elements of either <book> or <magazine> elements.

author[. = "Matthew Bob"]	All <author> elements whose value is Matthew Bob.
author[last-name = "Bob" and ../price > 50]	All <author> elements that contain a <last-name> child element whose value is Bob, and a <price> sibling element whose value is greater than 50.
book[position() <= 3]	The first three books (1, 2, 3).
author[not(last-name = "Bob")]	All <author> elements that do not contain <last-name> child elements with the value Bob.
author[first-name = "Bob"]	All <author> elements that have at least one <first-name> child with the value Bob.
author[* = "Bob"]	All author elements containing any child element whose value is Bob.
author[last-name = "Bob" and first-name = "Joe"]	All <author> elements that have a <last-name> child element with the value Bob and a <first-name> child element with the value Joe.

Expression	Refers to
./author	All <author> elements within the current context. Note that this is equivalent to the expression in the next row.
author	All <author> elements within the current context.
first.name	All <first.name> elements within the current context.
/bookstore	The document element (<bookstore>) of this document.
//author	All <author> elements in the document.
book[/bookstore/@specialty=@style]	All <book> elements whose style attribute value is equal to the specialty attribute value of the <bookstore> element at the root of the document.
author/first-name	All <first-name> elements that are children of an <author> element.
bookstore//title	All <title> elements one or more levels deep in the <bookstore> element (arbitrary descendants). Note that this is different from the expression in the next row.
bookstore/*/title	All <title> elements that are grandchildren of <bookstore> elements.
bookstore//book/excerpt//emph	All <emph> elements anywhere inside <excerpt> children of <book> elements, anywhere inside the <bookstore> element.
.//title	All <title> elements one or more levels deep in the current context. Note that this situation is essentially the only one in which the period notation is required.
author/*	All elements that are the children of <author> elements.
book/*/last-name	All <last-name> elements that are grandchildren of <book> elements.
/	All grandchildren elements of the current context.
***[@specialty]**	All elements with the specialty attribute.
@style	The style attribute of the current context.
price/@exchange	The exchange attribute on <price> elements within the current context.

Expression	Description
nodename	Sélectionne tous les nœuds avec le nom "nodename"
/	Sélectionne à partir du nœud racine
//	Sélectionne les nœuds du document du nœud actuel qui correspondent à la sélection, peu importe où ils se trouvent
.	Sélectionne le nœud actuel
..	Sélectionne le parent du nœud actuel
@	Sélectionne les attributs

XSL (XML Style Sheets Language / extension.xsl)

- Composé de trois parties :

XSLT : Langage de Transformation de documents XML

XPath : Langage de navigation dans les documents XML

XML-FO : Langage de Formatage des documents XML

- **XSLT :**

XSLT ==> XSL Transformation ==> XML StyleSheets Transformation.

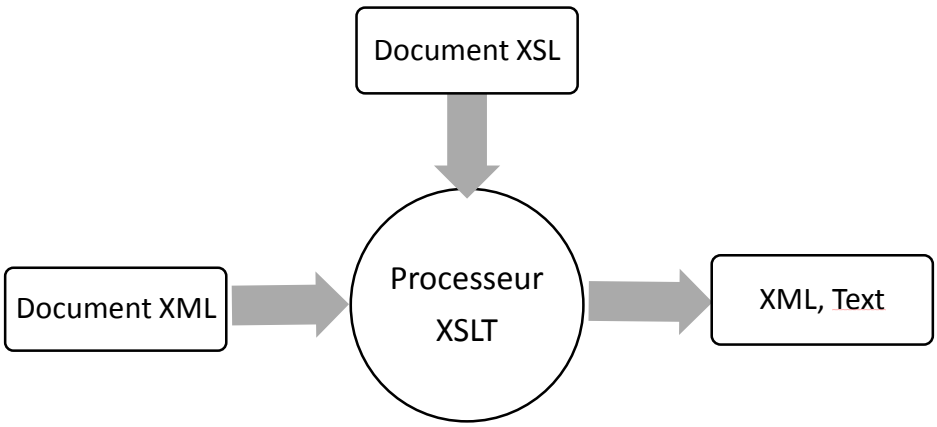
XSLT transforms an XML source-tree into an XML result-tree.

Permet de transformer un document XML en un autre document XML.

Une technologie qui permet de transformer les informations d'un document XML vers un autre type de document (XML, XHTML/HTML, CSV…).

Utilise le langage XPath pour naviguer dans un document XML (permet de sélectionner le (s) élément (s) sur lequel on veut appliquer un style.

Pour ça on utilise un **processeur** XSLT par exemple : Saxon.



XSL is a language for expressing style sheets. An XSL style sheet is, like with CSS, a file that describes how to display an XML document of a given type. XSL shares the functionality and is compatible with CSS2 (although it uses a different syntax). It also adds:

A transformation language for XML documents: XSLT. Originally intended to perform complex styling operations, like the generation of tables of contents and indexes, it is now used as a general purpose XML processing language. XSLT is thus widely used for purposes other than XSL, like generating HTML web pages from XML data.

Advanced styling features, expressed by an XML document type which defines a set of elements called Formatting Objects, and attributes (in part borrowed from CSS2 properties and adding more complex ones. (https://www.w3.org/Style/XSL/WhatIsXSL.html)

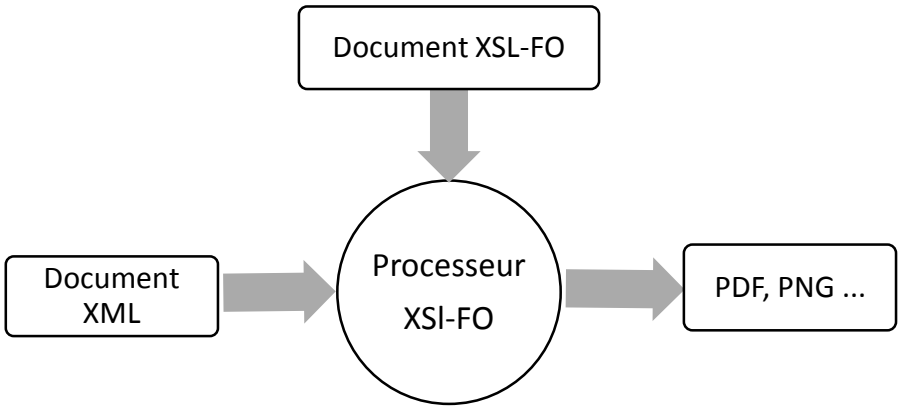
- **XSL-FO:**

= XSL Formatting Object.

Permet de formater un document XML en un autre document Imprimable.

Utilise le langage XPath et XSLT dans le processus de génération de documents.

Il est pour l'impression ce que CSS est pour les pages web.



- **Anatomie d'un fichier XSLT.**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">
    <xsl:output method="html" indent="yes"></xsl:output>

    <!-- template -->
</xsl:stylesheet>
<!-- meteo.xsl -->
```

XSLT output, qui se place sous la racine stylesheet, désigne la nature du document en sortie.

Indend = yes : la sortie va être indentée (tabulations d'imbrication, retour à la linge).

L'élément racine :

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

ou

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Les deux sont des synonymes.

- **Attacher une Feuille de Style XSL.**

```
<?xml-stylesheet type="text/xsl" href="meteo.xsl"?>
```

- **Templates (règles de production) :**

Un document xsl est un ensemble de templates (règles de traduction).

Le contenu d'un template permet de définir les transformations à appliquer à l'ensemble des données sélectionnées par l'expression XPath qui lui est attachée.

```
<xsl:template match="/">
```

L'attribut match reçois une expression XPath, dans ce cas, je dis que mon template va être appliqué sur tout le document XML.

Voici un exemple ou j'ai essayé de visualiser le contenu du fichier xml suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="meteo.xsl"?>
<meteo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="meteo.xsd">
    <mesure date="2006-1-1">
        <ville nom="Casa" temperature="22"/>
        <ville nom="Rabat" temperature="20"/>
        <ville nom="Fes" temperature="18"/>
        <ville nom="Oujda" temperature="19"/>
        <ville nom="Tanger"  temperature="25"/>
        <ville nom="Marrakech"  temperature="28"/>
        <ville nom="Ouarzazat"  temperature="29"/>
        <ville nom="Agadir"  temperature="20"/>
    </mesure>
    <mesure date="2006-1-2">
        <ville nom="Casa" temperature="21"/>
        <ville nom="Rabat" temperature="23"/>
        <ville nom="Fes" temperature="19"/>
        <ville nom="Oujda" temperature="20"/>
```

```
        <ville nom="Tanger"  temperature="23"/>
        <ville nom="Marrakech"  temperature="27"/>
        <ville nom="Ouarzazat"  temperature="25"/>
        <ville nom="Agadir"  temperature="23"/>
    </mesure>
</meteo>
```

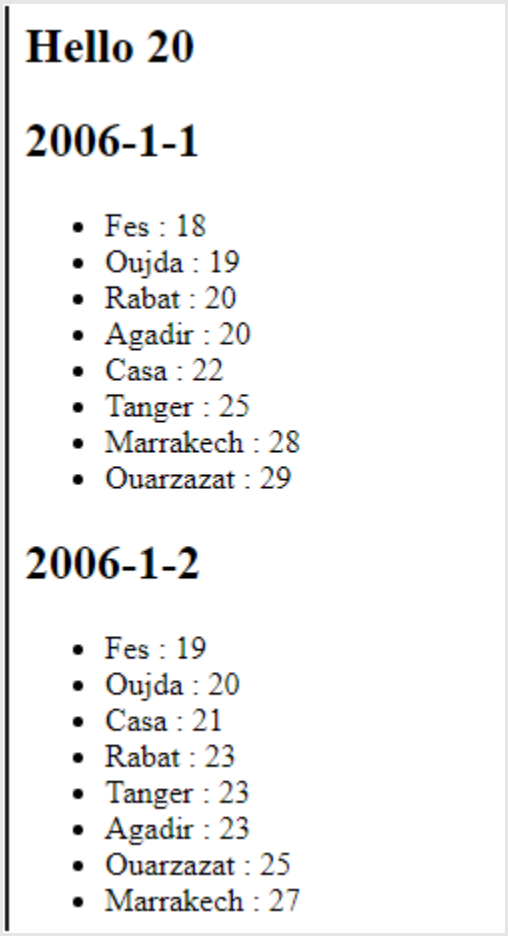
Voilà le fichier xsl.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">
    <xsl:output method="html" indent="yes"></xsl:output> <!-- use
indentation -->
    <xsl:template match="/">
        <html>
            <head>
                <title>XSLT</title>
            </head>
            <body>
                <h2>Hello  <xsl:value-of
select="meteo/mesure[1]/ville[@nom='Rabat']/@temperature"></xsl:value-
of> </h2>
                <!-- relative path in select parapport au element
selectionné dans le template -->

                <xsl:for-each select="meteo/mesure">
                    <h2>
                        <xsl:value-of select="@date"></xsl:value-of>
                    </h2>
                    <ul>
                        <xsl:for-each select="ville">
                            <xsl:sort select="@temperature"/>
                            <li>
                                <xsl:value-of
select="@nom"></xsl:value-of> : <xsl:value-of
select="@temperature"></xsl:value-of>

                            </li>
                        </xsl:for-each>
                    </ul>
                </xsl:for-each>

            </body>
        </html>
```

```
    </xsl:template>
</xsl:stylesheet>

<!-- meteo.xsl -->
```

Et voilà le résultat **html** :

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
      <title>XSLT</title>
   </head>
   <body>
      <h2>Hello  20</h2>
      <h2>2006-1-1</h2>
      <ul>
         <li>Fes : 18</li>
         <li>Oujda : 19</li>
         <li>Rabat : 20</li>
         <li>Agadir : 20</li>
         <li>Casa : 22</li>
         <li>Tanger : 25</li>
         <li>Marrakech : 28</li>
         <li>Ouarzazat : 29</li>
      </ul>
      <h2>2006-1-2</h2>
      <ul>
         <li>Fes : 19</li>
         <li>Oujda : 20</li>
         <li>Casa : 21</li>
         <li>Rabat : 23</li>
         <li>Tanger : 23</li>
         <li>Agadir : 23</li>
         <li>Ouarzazat : 25</li>
         <li>Marrakech : 27</li>
      </ul>
   </body>
</html>
```



- A template contains rules to apply when a specified node is matched.
- The <xsl:template> element is used to build templates.
- The match attribute is used to associate a template with an XML element.

- The match attribute can also be used to define a template for the entire XML document. (i.e. match="/" defines the whole document).
- **Le expressions XPath utilisés, doivent être relatifs au '/' attaché au template principale, et donc commencer avec l'élément root du fichier XML sans le slash '/'.**
- **Les fonctions XSL :**

Value-of : Extraire la valeur d'un élément XML ou la valeur de ses attributs.	`<xsl:value-of select="meteo/mesure[1]/ville[@nom='Rabat']/@temperature"/>`

For-each:
Créer une boucle sur un ensemble d'éléments sélectionné
	`<xsl:for-each select="ville">`
	`  <xsl:sort select="@temperature"/>`
	`  <li>`
	`  <xsl:value-of select="@nom"/>`
	`  </li>`
	`</xsl:for-each>`

Sort : Trier en ordre croissant ou décroissant	Généralement utilisée au sein d'une fonction <xsl:for-each /> (voir l'exemple de for-each). - autres attributs optionnels : **Order**: ascending (croissant), descending (décroissant) **Case-order**: upper-first (les majuscules d'abord), lower-first (les minuscules d'abord) **Data-type** : text (texte) et number (numérique) **Lang** : code d'une langue (fr, en, it, etc.)

If :
Conditionner une transformation
	Opérateurs logiques : AND, OR, NOT
	Opérateurs de comparaison : = , not (a = b), < pour inférieur, > pour supérieur
	
	`<xsl:if test="@temperature > 20">`
	`  <xsl:text > ++ </xsl:text>`
	`</xsl:if>`

Choose : (== switch case)
Traiter des cas (plusieurs conditions)
	`<xsl:choose>`
	`  <xsl:when test="@temperature > 15 and @temperature < 27">`
	`  (il fait Beau)`
	`  </xsl:when>`
	`  <xsl:when test="@temperature > 27">`
	`  (il fait chaud)`
	`  </xsl:when>`
	`  <xsl:otherwise>`
	`  (il fait froid)`
	`</xsl:otherwise>`
	`</xsl:choose>`

<table>
<tr>
<td>apply-templates :
permet de continuer la transformation des éléments enfants d'un template</td>
<td>Un simple apply-templates (sans attribut select) examine tous les nœuds enfants dans l'ordre. Si une règle qui correspond à un nœud est détectée, elle sera appliquée.
[Applies a template rule to the current element or to the current element's child nodes]
<pre>
<xsl:template match="/">
 <html>
   <head>
     <title>XSLT</title>
   </head>
   <body>
    <xsl:for-each select="meteo/mesure">
      <h2>
       <xsl:apply-templates select="."/>
      </h2>
      </xsl:for-each>
   </body>
  </html>
 </xsl:template>
 <xsl:template match="mesure">
       Date : <xsl:value-of select="@date"/>
       Nombre villes:<xsl:value-of
select="count(ville)"/>
 </xsl:template>
</pre>
</td>
</tr>
<tr>
<td>Variable
Declares a local or global variable</td>
<td>
- The variable is global if it's declared as a top-level element, and local if it's declared within a template.
- Once you have set a variable's value, you cannot change or modify that value!
- You can add a value to a variable by the content of the <xsl:variable> element OR by the select attribute!
<pre>
<xsl:variable name="header">
  <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
  </tr>
</xsl:variable>

<xsl:template match="/">
  <html>
  <body>
  <table border="1">
    <xsl:copy-of select="$header" />
    <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
</pre>
</td>
</tr>
</table>

	`</tr>`
	`</xsl:for-each>`
	`</table>`
	`</body>`
	`</html>`
	`</xsl:template>`

Apply-imports	Applies a template rule from an imported style sheet
Attribute	Adds an attribute
Comment	Creates a comment node in the result tree
attribute-set	Defines a named set of attributes
call-template	Calls a named template
copy	Creates a copy of the current node (without child nodes and attributes)
copy-of	Creates a copy of the current node (with child nodes and attributes)
element	Creates an element node in the output document
text	Writes literal text to the output