XML :

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

Elément avec attribut

<nom_element nom_attribut="…"> </nom_element>

Espace de nom :

- Par défaut : <nom_element xmlns= "….">
- Explicite : <allias :nom-element xmlns :allias = "…"

DTD :

- **Interne** : <!DOCTYPE nom_element […]>
- **Externe** : <!DOCTYPE nom_element SYSTEM "nom-dtd.dtd">

Définition d'un élément

<!ELEMENT nomElement CONTENU>

- Contenu:
  - Any, Empty, #PCDATA
  - Séquence d'éléments : element1,element2,..
  - Choix d'éléments: element1 | element2 …

Cardinalité d'éléments :

- * : 0 ou plusieurs, + : 1 ou plusieurs, ?: 0 ou 1

Les attributs :

<!ATTLIST nom_élément
nom_attribut TYPE OBLIGATION
VALEUR_PAR_DEFAUT>

- TYPE :
  - CDATA, ID, IDREF, enumeration (valeur1 | valeur2..)
- OBLIGATION :
  - #REQUIRED(Obligatoire), #IMPLIED (Optionnel), #FIXED(valeur fixe)

XML schema

Assignation à un document xml

<nom_element
xmlns :xs="http://www.w3.org/2001/XMLSchema-instance"
xs :noNamespaceSchemaLocation="nom_fichier_xmlSchema.xsd">

Types simples :

- *Type simple de type Liste :*

```
<xs :simpleType name="nom_type">
    <xs :list itemType="type-utilisé"/>
</xs :simpleType>
```

- *Type simple par union*

```
<xs :simpleType name="nom_type">
    <xs :union memberTypes="type1 type2…"/>
</xs :simpleType>
```

Types complexes

- *Séquence :*

```
<xs :complexType name="nom_type">
    <xs :sequence>  …  </xs :sequence>
</xs :complexType >
```

- *Choix :*

```
<xs:complexType … >
    <xs:choice> …  </xs:choice>
</xs:complexType>
```

- *All* :

```
<xs:complexType … >
    <xs:all> … </xs:all>
</xs:complexType>
```

Définition d'un élément :

- <xs :element name="nom_element" type="le_type"/>

Les cardinalités des éléments:

- minOccurs/maxOccurs, unbounded

```
<xs :element name="nom_element" type="le_type"
 minOccurs ="…" maxOccurs="…"/>
```

Type avec contenu mixte :

```
<xs:complexType name="nom_type" mixed="true">
  ...
</xs:complexType>
```

Les attributs:

```
<xs:complexType…>
    …
   <xs:attributes name="nom" use ="required |
    optional|prohibited type="le_type" />
</xs:complexType… >
```

Extension des types :

-

```
<xs:complexType  …>
    <xs:simpleContent>
       <xs :extension base="type-de-base">
                ….
       </xs :extension>
    </xs:simpleContent>
</xs:complexType>
```

-

```
<xs:complexType …>
    <xs:complexContent>
       <xs :extension base="type-de-base">
       ….
       </xs :extension>
    </xs:complexContent>
</xs:complexType>
```

Restriction des types :

- *Type simple par intervalle :* minExclusive, maxExclusive, minInclusive, maxExclusive :

```
<xs :simpleType name="nom_type">
    <xs :restriction base="type-de-base">
       <xs:minInclusive value= … />
        <xs:maxInclusive value= … />
    </xs :restriction>
</xs :simpleType>
```

- *Type simple par énumération :*

```
<xs :simpleType name="nom_type">
    <xs :restriction base="type-de-base">
       <xs:enumeration value ="…"/>
        <xs:enumeration value ="…"/>
       ...
    </xs :restriction>
</xs :simpleType>
```

- *Type simple par motif :*

```
<xs :simpleType name="nom_type">
    <xs :restriction base="type-de-base">
      <xs:pattern  value ="…"/>
      ….
    </xs :restriction>
</xs :simpleType>
```

- *Types complexes :*
  - ```
    <xs:complexType …>
      <xs:simpleContent>
        <xs:restriction base= "type-de-base">
          …
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
    ```
  - ```
    <xs:complexType …>
      <xs:complexContent>
        <xs:restriction base= …>
          …
        </xs:restriction>
      </xs:complexContent>
    </xs:complexType>
    ```

Unicité :

```
<xs:unique name="e">
      <xs:selector xpath="l'élément concerné"/>
      <xs:field xpath= "partie concernée"/>
</xs:unique>
```

Clé: Comme pour l'unicité en remplaçant **xs :unique** par **xs:key**

Référence :

```
<xs:keyref name=  "nomReference" refer= "nom_Clé">
    <xs:selector xpath="element"/>
    <xs:field xpath="partie_concernée"/>
</xs:keyref>
```

XPath : Axe ::filtre[predicat]*

Les axes : ancestor, ancestor-or-self, child, descendant, desendant-or-self, parent, following-sibling, following, preceding, preceding-sibling, attributes

Les filtres: nom_element, *, comment(), text(),…

Expression abrégée: / : child, // : descendant, . :self, .. : parent, @ : attribute

Union : expression_xpath1 **union** expression_xpath2
Intersection: expression_xpath1 **intersect** expression_xpath2
Différence: expression_xpath1 **except** expression_xpath2

XSLT :

Définition d'une feuille XSL

```
<?xml version =" 1.0 " encoding=" iso-8859-1" ?>
    <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     ...
    </xsl:stylesheet>
```

Association d'une feuille xsl à un document XML

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

Définition d'un template

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

Application d'un template:

```
<xsl:apply-templates select="…"/>
```

Appel explicite de template :

```
<xsl:call-templates name="nom_de_template"/>
```

Récupération de contenu :

```
<xsl:value-of select="nom-du-noeud"/>
```

Structures conditionnelles :

- ```
  <xs:if test ="expression1">expression2 </xsl:if>
  ```
- ```
  <xs :choose>
    <xs:when test=…> … </xs:when>
    <xs :otherwise>…</xs :otherwise>
  </xs :choose>
  ```

Boucle : <xsl :for-each select="…">…</xsl :for-each>
Tri : <xsl :sort select ="…"/>

XQuery:

doc("chemin_document_xml.xml") : retourne l'arbe correspondant au document xml
collection("nom_collection") : retourne une forêt XML

Expression FLOWER (For, let, where, order by, return) :

Expression let:
Let $variable := expression
Where condition
Return collection

Expression for :
For $variable in expression
Where condition
Return collection

Expression conditionnelle :
if (condition) then {expression}
else {expression}

Ordonner :
For $variable in expression
Where condition
Return collection
Sort by (élément d'ordonnancement)

Constructeurs :
element {nom-element}{valeur-element}
attribute {nom-attribute}{valeur-attribute}