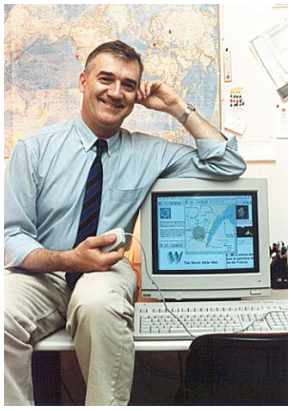
Table des matières


Robert Cailliau

Histoire

Début des années 80 **Charles Goldfarb** de la société IBM a été à l'origine du *SGML* :Standardized generalized markup languagage (finalisé en 1986 norme iso:8879)
En 1989 **Tim Berners-lee** et **Robert Cailliau** créent le www.

Le HTML est né


Berners-Lee, 2005

XML : eXtensible Markup Language

Voir http://svground.fr//tutorial-xml.php

Le XML, en lui-même, ne fait rien ! le XML n'est pas un langage mais un métalangage, c'est à dire une grammaire qui sert à définir des langages.
Alors que le Html a été conçu pour afficher de l'information, le XML a été créé pour structurer de l'information. Il ne fait rien d'autre !
L'objectif initial de XML est expliqué au début de la spécification du 10 février 1998, la phrase est toujours d'actualité : « Son but est de permettre au SGML générique d'être transmis, reçu et traité sur le web de la même manière que l'est HTML aujourd'hui. »

Voici un exemple de XML.

```
<?xml version="1.0"?>
<demoXML>
        <message>Voici du XML</message>
</demoXML>
```

Ce qui affiché dans le Internet Explorer donne le résultat suivant.

```
<?xml version="1.0" ?>
- <demoXML>
    <message>Voici du XML</message>
  </demoXML>
```

Pas que quoi s'extasier !.. Le XML n'est que de l'information encodée entre des balises. Il faudra d'autres éléments pour que le navigateur puisse "comprendre" vos balises et afficher ce fichier sous une forme plus conviviale.
il existe trois solutions pour mettre en forme un document XML :

- CSS (*Cascading StyleSheet*), la solution la plus utilisée actuellement, étant donné qu'il s'agit d'un standard qui a déjà fait ses preuves avec HTML
- XSL (*eXtensible StyleSheet Language*), un langage de feuilles de style extensible développé spécialement pour XML.
- XSLT (*eXtensible StyleSheet Language Transformation*). Il s'agit d'une recommandation W3C du 16 novembre 1999, permettant de transformer un document XML en document

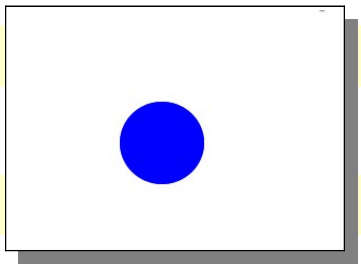
Exemple: dessin vectoriel svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100" height="100" version="1.1"
 xmlns="http://www.w3.org/2000/svg">

    <circle cx="50" cy="50" r="40"  style="fill:#00F"/>

</svg>
```



Structure d'une simple page SVG

a. Prologue

Un fichier SVG commence par une déclaration de version XML standard, comme par exemple

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

Il faut alors ajouter le DocType correspondant à la version SVG utilisée :

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

b. Élément racine

La racine d'un fichier SVG est un élément...< **svg**>. Mais il est nécessaire de définir deux espaces de noms, un par défaut et un second permettant d'avoir accès à d'autres fonctionnalités comme suit :

<svg xmlns="http://www.w3.org/2000/svg" xmlns:link="http://www.w3.org/1999/xlink">(..)
</svg>

La taille de la fenêtre SVG est définie par les attributs width et height de l'élément <svg> : <svg width="400" height="250" xmlns="http://www.w3.org/2000/svg">

c. Le contenu

```
 <rect
     width="100" height="80"
     x="0" y="70"
     fill="green" />
  <line
     x1="5" y1="5"
     x2="250" y2="195"
     stroke="red" />
  <circle
     cx="90" cy="80"
     r="50"
     fill="blue" />
   <text x="170" y="50">
     Bonjour à tous
   </text>
```

XML n'a pas que des avantages...

Tout d'abord, XML est très verbeux. C'est à dire qu'il utilise beaucoup de caractères.
Par exemple, si on veut noter le nombre d'exemplaires, on ferait sûrement:
<NOMBREEXEMPLAIRES>5</NOMBREEXEMPLAIRES>
L'information elle-même n'occupe qu'un octet ("5"), mais on l'a encadré de 39 octets.
C'est un énorme gaspillage de place.
Ensuite, ce format étant en texte, l'ordinateur est obligé de tout convertir en binaire (pour s'en faire une "représentation" en mémoire), faire ses traitements, puis tout reconvertir en texte dans l'autre sens. Cela prend beaucoup de temps (surtout quand on a beaucoup d'XML à traiter).
Ensuite, l'XML est très bien adapté à la représentation d'informations hiérarchiques, mais que des informations hiérarchiques.
Il est très mal adapté à la représentation d'autres types de données, comme les données relationnelles.
Par exemple: Un catalogue contient plusieurs produit; ces produits ont été commandés dans telle et telle commande; cette commande vient de tel client; etc. XML est mal adapté au stockage de telles informations, au contraire des bases de données relationnelles.
Dans la vraie vie, toutes les informations n'entrent pas dans de petites catégories bien hiérarchisées.
L'XML est également incapable de stocker des informations binaires, comme des images, vidéos ou musiques.
Il existe bien une astuce pour représenter sous forme de texte des données binaires (l'encodage en base64), mais on perd énormément de place. (Un fichier de 100 000 octets prendra 136 848 octets en base64 !)
Et enfin, dans la pratique, les fichiers XML qu'on rencontre sont fortement imbriqués, ce qui rend la lecture par un humain quasi-impossible.
De plus, les technologie qui tournent autour d'XML (XSD, XSLT, XPath...) sont nombreuses et complexes: il est difficile de bien les maîtriser.
Certaines ne sont pas adoptées par tout le monde, ou sont poussées par des entreprises pour leur propre intérêt.
Donc l'XML n'est pas la panacée.
XML n'est que l'une des nombreuses manières de stocker et transmettre de l'information.

Où retrouve-t-on de l'XML ?

On en retrouve un peu partout.

On en retrouve dans les pages web:

l'XHTML (validé par le W3C) est le successeur d'HTML. XHTML a la particularité d'être du vrai XML (ce qui n'est pas le cas d'HTML 4 ou 5).

Les documents OpenOffice

(.dot par exemple) sont également des fichiers XML, zippés dans un même fichier.
Certains logiciels de chat comme Jabber utilisent XML pour échanger les messages.
Le format graphique SVG permet de définir des formes géométriques. La plupart des navigateurs récents sont capables de les afficher dans les pages web. C'est également le format utilisé par des logiciels de dessin comme InkScape .

MathML

permet de décrire et échanger des formules mathématiques.
SMIL est un format conçu pour les présentations multimédia. C'est également du XML.

RSS permet aux sites web de publier leurs informations. Les logiciels comprenant RSS sont capables de récupérer les infos des sites qui vous intéressent, les aggréger et vous les présenter. RSS utilise un schéma XML particulier.

Java (applets et programmes) et Microsoft .Net (inclu dans Windows XP)

utilisent des fichiers XML pour décrire les programmes et leurs paramètres de sécurité.
XML-RPC et SOAP permettent à des programmes d'appeller des fonctions d'autres programmes à travers internet: c'est ce qu'on appelle les webservices. XML-RPC et SOAP utilisent exclusivement XML.
Des systèmes comme Rosetta.Net ou UBL (Universal Business Language) sont conçus pour permettre aux entreprise d'échanger des commandes, des factures, des devis...
Le logiciel FreeMind utilise le format XML pour sauvegarder (fichiers .mm).
Beaucoup de logiciels utilisent le format XML pour stocker leur configuration.

La norme XML

La norme XML en tant que telle doit être vue comme un outil permettant de définir un langage (on dit alors qu'il s'agit d'un **métalangage**), permettant de créer des documents structurés à l'aide de balises.
Une balise est une chaîne de caractère du type:

<balise> et normalement </balise>

Ainsi, un document XML, c'est-à-dire le fichier créé en suivant les spécifications de la norme XML pourra ressembler à ceci:

```
<?xml version="1.0" encoding="UTF-8"?>
<annuaire>
      <personne class = "gestionnaire">
            <nom>Kaisse</nom>
            <prenom>Jean</prenom>
            <telephone>04-123456</telephone>
            <email>comptable@site.com</email>
      </personne>
      <personne>
            ...
      </personne>
</annuaire>
```

D'un point de vue formel, un document XML est un arbre.
Enfin il est possible d'ajouter des commentaires dans le document XML de la manière suivante:
<!-- Voici des commentaires XML -->
mais pas <!----------> pas plus de deux tirets à chaque extrémité !!!!!

Remaque générale sur la casse

N'oubliez pas que XML est sensible à la casse, par conséquent `<tag>` et `<Tag>` n'ont pas la même signification.

Structure d'un document XML

En réalité un document XML est structuré en 3 parties:

1. **prologue** XML
2. une déclaration de **type de document**
3. **arbre**

La première partie:Déclaration XML - prologue XML

Tout document XML devrait commencer par une ligne de code telle que :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
```

Cette ligne, appelée aussi prologue XML, définit :

1. La version de XML .C'est le minimum requis pour la déclaration XML :
   <?xml version="1.0" ?>
2. Le type d'encodage .Cette ligne permet donc d'indiquer la version XML utilisée, le jeu de caractères utilisé et l'autonomie du document.
3. Le dernier attribut, standalone .Il prend pour valeur **yes** ou **no**, indique si oui ou non, une **DTD explicite** accompagne le document XML; par défaut, c'est non : ce qui signifie que la DTD est la l**a grammaire du langage** déclarée avec une instruction <!DOCTYPE>. (on y reviendra)

Notez que dans un fichier XHTML, ce prologue XML est souvent omis parce qu'il génère des incompatibilités CSS avec MSIE 6. Cependant tout se passera comme si l'instruction XML suivante

était présente :

<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>

Ce qui signifie qu'il faut associer le fichier XML/XHTML à une DTD.
Les attributs encoding et standalone sont facultatifs.
L'ordre des attributs est important.
Par défaut, la norme d'encodage est **utf-8**

UTF-8 :

Format international qui permet d'écrire dans n'importe quelle langue. Chaque caractère à un codage unique (Unicode). Les caractères latins (a-z et A-Z), chiffres arabes (0-9) et quelques autres caractères de ponctuation française sont codés sur un seul octet. Les autres caractères sont codés sur plusieurs octets (la taille peut varier de 2 à 4 caractères, peut-être plus ?). Les caractères accentués français prennent par exemple deux octets.

Jeux de caractères Liste non exhaustive	
Norme	Description
UTF-8	Jeu de caractères universel sur 8 bits.
UTF-16	Jeu de caractères universel sur 16 bits.
ISO-8859-1	Latin 1 - Langues d'Europe de l'ouest et d'Amérique latine.
ISO-8859-2	Latin 2 - Langues d'Europe centrale et Slaves.
ISO-8859-3	Latin 3 - Langues Epéranto, Galicienne, Maltaise et Turc.
ISO-8859-4	Latin 4 - Langues Estonienne, Lettone et Lithuanienne.
ISO-8859-5	Langue Cyrilliques.
ISO-8859-6	Langue Arabe.
ISO-8859-7	Langue Grecque.
ISO-8859-8	Langue Hébraïque.
ISO-8859-9	Latin 5 - Langue Turc.
ISO-8859-10	Latin 6 - Langues Groenlandaises et Lapones.

Caractères spéciaux

Certains caractères ayant une signification particulière dans la grammaire du XML restent interdits : <, >, &.
Pour ces caractères comme pour les caractères pouvant poser des problèmes à l'affichage, utilisez les caractères de masquage (d'échappement) soit sous forme d'entités nommées, soit sous forme d'entités codées.

Caractères de masquage Liste non exhaustive		
Caractère	Entité nommée	Entité codée
<	<	<
>	>	>
&	&	&

Une autre alternative consiste à préciser à l'analyseur XML de ne pas interpréter ces caractères spéciaux pour le texte selectionné.
Pour cela, vous devez préciser le texte entre **<![CDATA[** et **]]>**.

```
<?xmlversion="1.0"encoding="ISO-8859-1"?>
<annuaire>
<societe>
<nom><![CDATA[GarageDupond&Fils]]></nom>
<description><![CDATA[Distance du centreville<2km]]></description>
</societe>
</annuaire>
```

L'exemple suivant (en XHTML) n'est pas très lisible :

```
<p>Hier j'ai écrit un fichier XML de ouf !</p>
<p>&lt;monfichierXML&gt;
    &lt;balise1&gt;
        &lt;balise2&gt;Du texte&lt;/balise2&gt;
    &lt;/balise1&gt;
    &lt;balise1 truc=&quot;pouf&quot; paf=&quot;pif&quot;/&gt;
&lt;/monfichierXML&gt;</p>
```

tandis qu'avec une section CDATA, plus aucun problème :

```
<p>Hier j'ai écrit un fichier XML de ouf !</p>
<p><![CDATA[<monfichierXML>
    <balise1>
        <balise2>Du texte</balise2>
    </balise1>
```

```
    <balise1 truc="pouf" paf="pif"/>
</monfichierXML>]]></p>
```

Le code entre <![CDATA[et]]> n'est pas interprété, c'est-à-dire qu'il est considéré comme du texte, pas comme du XML. La chaîne de caractères «]]> » est interdite dans une section CDATA (logique, puisque cette séquence de caractères indique la fin d'une section CDATA). Une section CDATA ne peut donc pas contenir de section CDATA (mais qui aurait une idée aussi tordue :D ?).

Deuxième partie : une déclaration de type de document

c'est une déclaration de type de document (à l'aide d'un fichier annexe appelé DTD - Document Type Definition)

Exemple : xhtml 5

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr" dir="ltr">
  <head>
    <meta http-equiv="Content-Type" content="application/xhtml+xml;
     charset=UTF-8" />
    <title>Exemple</title>
  </head>
   <body>
     <!-- Contenu de la page respectant la syntaxe XML. -->
   </body>
</html>
```

l'arbre des éléments

L'élément racine

L'élément racine est lui aussi obligatoire. Cet élément est une balise crée par vos soins. Elle est unique dans le document. Elle peut être comparée à la balise `<body></body>` d'un document HTML. Elle encadrera le contenu de votre document XML.

L'élément racine est obligatoire tout comme le prologue. Cet élément doit être unique dans le document.

Exemple, l'élément racine *librairie* :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<librairie>
...
</librairie>
```

L'arbre des éléments

L'arbre des éléments est constitué d'une hiérarchie de balises comportant éventuellement des attributs.
Un attribut est une paire **Nom_atribut valeur** écrit sous la forme *Nom_atribut="Valeur"*, ainsi une

balise affectée d'un attribut aura la syntaxe suivante:

<balise Nom_atribut="valeur">
Toute donnée est ainsi encapsulée entre une balise ouvrante <*balise*> et une balise fermante </*balise*> .

```
<bag couleur="bleu" marque="decathlon">
  Mon sac de sport.
</bag>
```

Les attributs

Les valeurs des attributs doivent être précisés, au choix, entre guillemets " ou entre apostrophes '.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<racine>
    <mabalise monattribut1="valeur1" />
    <mabalise monatrribut2='valeur2' />
</racine>
```

Une valeur précisée entre apostrophes peut contenir des guillemets, une valeur précisée entre guillemets peut contenir des apostrophes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<racine>
    <mabalise monattribut1='il a dit "Bonjour!"' />
    <mabalise monattribut2="c'était aujourd'hui" />
</racine>
```

Si vous souhaitez indiquer une valeur d'attribut contenant des guillemets (alors que celle-ci est délimitée par des guillemets) vous devrez utiliser la notation
" ou le code ASCII **"**.
Si vous souhaitez indiquer une valeur d'attribut contenant des apostrophes (alors que celle-ci est délimitée par des apostrophes) vous devrez utiliser la notation
' ou le code ASCII **'**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<racine>
    <mabalise monattribut="il a dit &quot;Bonjour!&quot;" />
    <mabalise monatrribut='c&apos;était aujourd&apos;hui' />
</racine>
```

Élément vide

Un **élément vide** est uniquement constitué d'une balise spécifique dont la syntaxe est la suivante : ***<balise />***.
exemples
 <img /> <input />

```
exemple: <img src="image.png" />
```

Remarque:*Une balise vide peut s'écrire de différents manières. Prenons l'exemple de la balise HTML br. On peut l'écrire sous trois formes différentes :*

```
<br/>
<br />
<br></br>
```

il est interdit en XML de faire chevaucher des balises, c'est-à-dire d'avoir une succession de balises du type:

```
<balise1>
        <balise2>
</balise1>
        </balise2>
```

D'autre part, il est possible, entre les balises (donc pas à l'intérieur d'une balise), d'ajouter:

- des espaces
- des tabulations
- des retours chariots

Cependant, cela obligera à quelques manipulation dans certains analyseurs qui tiennent compte de ces caractères supplémentaires.
Voici un document XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire>
   <personne type="étudiant">
      <nom>FAIR</nom>
      <prenom>Lucie</prenom>
      <email>webmaster@satanas.com</email>
   </personne>
   <personne type="chanteur">
      <nom>Gueule</nom>
      <prenom>Jean</prenom>
      <email>penible@aentendre.seul</email>
   </personne>
</annuaire>
```

Espace de nom

xmlns signifie **xml n**ame **s**pace

Espaces de noms et langue du document

Les espaces de noms sont une manière d'assurer que les noms d'éléments utilisés par le document XML n'entreront pas en conflit avec les noms d'autres documents qui pourraient être de type similaire et combinés entre eux.

XHTML :

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="fr" lang="fr" dir="ltr">
```

L'attribut `xmlns`, pour XML Naming Space, indique une URL unique qui identifie de manière unique les éléments de XHTML par rapport à d'autres éléments qui pourrait entrer en conflit. Cet attribut apparaît dans le marqueur d'ouverture de l'élément racine du document XML, ici la balise `<html>`.

Enfin, l'attribut `xml:lang` permet de spécifier la langue relative à l'élément XML concerné. Le code est composé de deux lettres minuscules qui se rapportent à un code langue, d'un tiret et de deux lettres majuscules correspondant à un code pays.

Voir :http://zvon.developpez.com/tutoriels/dtd/http://zvon.developpez.com/tutoriels/dtd/

Exemples concrets:

Une adresse

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personne>
  <nom>Jean Nemard</nom>
```

```
  <email>JeanNemard SUR Serveur.org</email>
  <adresse> 10, rue de la poule, Liege(Belgique)</adresse>
</personne>
```

Une représentation plus précise:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personne>
  <nom>
    <prenom>Jean</prenom>
    <nom>Nemard</nom>
  </nom>
  <email>
    <identifiant>JeanNemard</identifiant>
    <serveur>Serveur.org</serveur>
  </email>
  <adresse>
    <numero>10</numero>
    <rue>rue de la poule</rue>
    <ville>Liège</ville>
    <pays>Belgique</pays>
  </adresse>
</personne>
```

En utilisant les attributs:

```
<personne>
  <nom>
  <prenom valeur= "Jean" />
  <nom valeur="Nemard" />
  </nom>
  <email>
    <identifiant valeur="JeanNemard" />
    <serveur valeur="Serveur.org" />
  </email>
  <adresse>
    <numero>10</numero>
    <rue>rue de la poule</rue>
    <ville>Liège</ville>
    <pays>Belgique</pays>
  </adresse>
</personne>
```

parseur

Pour visualiser un document XML, il vous faut un **"parser"** XML (parseur en français !?).

Un parseur est un analyseur convertisseur syntaxique.

Ajouter une simple feuille de style

Cela change tout !!

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

Exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="bibliotheque.css" ?>
<bibliotheque >
    <roman>
        <titre>imajica</titre>
        <auteur>clive barker</auteur>
        <prix>6</prix>
    </roman>
    <roman>
        <titre>Dune</titre>
        <auteur>Frank Herbert</auteur>
        <prix>7</prix>
    </roman>
    <magazine>
        <titre>science et vie</titre>
        <dateparution>2005-02-01</dateparution>
    </magazine>
    <roman>
        <titre>christine</titre>
        <auteur>stephen king</auteur>
        <prix>5</prix>
    </roman>
</bibliotheque>
```

et sa feuille de style:

```
* {text-align:center}
bilioteheque ,roman,magazine,titre {display:block;}
auteur, prix,dateparution {display:inline}
bibliotheque {margin-top:10px;}
roman,magazine {width:50%;padding-left:20px;border:solid 1px}
roman,magazine {margin:auto}
roman,magazine,titre {text-align:left}
titre {color:blue; font-size:25pt}
prix {font-style:italic}
dateparution {color:#95704A}
magazine titre {color:green}
/*  -firefox IE9-  css2  */
dateparution:before {content:" Date de parution: "}
prix:before {content:" prix: "}
prix:after {content:" € "}
auteur:before {content:"Auteur: "}
```

Instructions de traitement et section CDATA

- Instructions de traitement :
Les instructions de traitement permettent de fournir des informations supplémentaires sur le document aux analyseurs syntaxiques.
Une instruction de traitement commence par <? et se termine par ?>.
La plus utilisée de ces instructions et sûrement celle constituant le prologue d'un document XML :
`<?xml version="1.0"?>`
L'inclusion d'une référence à une feuille de styles utilise aussi ce type d'instruction. Nous en reparlerons le moment venu.

- Section CDATA :
Cette section permet d'inclure des données textuelles, des exemples de code, ... dans un document XML sans qu'il soit nécessaire de substituer les caractères spéciaux par des caractères de masquage. La mise en œuvre d'une section CDATA s'écrit ainsi :
`<![CDATA[`
données textuelles
`]]>`

Les sections CDATA ne peuvent pas être imbriquées.

Exemple.svg horloge

```
<svg onload='Init(evt)' width='124' height='124' viewBox='0 0 124 124'

     xmlns:xlink='http://www.w3.org/1999/xlink'
     xmlns='http://www.w3.org/2000/svg'
     xmlns:a3="http://ns.adobe.com/AdobeSVGViewerExtensions/3.0/"
    a3:scriptImplementation="Adobe"
     >
  <title>Une horloge en SVG</title>

  <defs>
    <script><![CDATA[
        function SetTime () {
            var Now = new Date();

            var Seconds = Now.getSeconds();
            var Minutes = Now.getMinutes() + Seconds / 60;
            var Hours = Now.getHours() + Minutes / 60;
            var Year = Now.getFullYear();

            sec.setAttributeNS(null,   'transform', 'rotate(' + (Seconds * 6) +
')');
            min.setAttributeNS(null,   'transform', 'rotate(' + (Minutes * 6) +
')');
            hours.setAttributeNS(null, 'transform', 'rotate(' + (Hours * 30)  +
')');
            day.firstChild.data = Now.getDate()+ '/' + (Now.getMonth()+1) + '/' +
Year;
          }

        function Init (LoadEvent) {
          // globals
           SVGDocument = LoadEvent.target.ownerDocument;
          sec = SVGDocument.getElementById('seconds');
          min = SVGDocument.getElementById('minutes');
```

```
          hours = SVGDocument.getElementById('hours');
          day = SVGDocument.getElementById('day');

          SetTime();
                              window.SetTime = SetTime;
                              setInterval('window.SetTime()', 1000);
        }
    ]]>
</script>

    <path id='outcircle' d='M -46 0 C -46 -63 46 -63 46 0'/>
  </defs>
  <rect fill='#3e5255' width='124' height='124'/>

  <g transform='translate(0,5)'>
    <g transform='translate(62 56)'>
      <g>
        <g fill='white' stroke='white'>
          <circle cx='0' cy='-40' r='3'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(30)'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(60)'/>
          <circle cx='0' cy='-40' r='3' transform='rotate(90)'/>

          <circle cx='0' cy='-40' r='2' transform='rotate(120)'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(150)'/>
          <circle cx='0' cy='-40' r='3' transform='rotate(180)'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(210)'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(240)'/>
          <circle cx='0' cy='-40' r='3' transform='rotate(270)'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(300)'/>
          <circle cx='0' cy='-40' r='2' transform='rotate(330)'/>
        </g>

        <text font-family='Arial, Helvetica, sans-serif' font-size='10pt' font-
weight='bold' fill='#8fa391'>
          <textPath xlink:href='#outcircle' letter-spacing='4'><a
xlink:href='http://xmlfr.org/index/object.title/SVG/'>Réalisé en
SVG</a></textPath>
        </text>
      </g>

      <g id='hours' stroke-width='2' stroke='#aa1717' fill='#aa1717'>
        <circle cx='0' cy='-30' r='3'/>
        <line x1='0' y1='0' x2='0' y2='-30'/>
      </g>

      <g id='minutes' stroke-width='2' stroke='#aa1717' fill='#aa1717'>
        <circle cx='0' cy='-40' r='3'/>
        <line x1='0' y1='0' x2='0' y2='-40'/>
      </g>
      <g id='seconds'>
        <line x1='0' y1='10' x2='0' y2='-50' stroke-width='1' stroke='white'/>
      </g>

      <text id='day' x='0' y='57' text-anchor='middle' fill='white' font-
family='Verdana, Arial, Helvetica, sans-serif'>01/01/2000</text>

      <circle cx='0' cy='0' r='3' fill='white' stroke='white'/>
    </g>
  </g>
</svg>
```

Document valide et bien formé

Un document est appelé "document XML" s'il est

bien formé,

signifie qu'il respecte les règles syntaxiques de XML.

valide

si, en plus, il respecte l**a grammaire du langage**, contenue dans un fichier appelé DTD (Définition de Type de Document).

Il y a quatre types de déclarations:

- éléments
- listes d'attributs
- entités
- **notations**

DTD

Un document XML est valide s'il est associé à une **définition de type de document** et s'il respecte les contraintes qui y sont définies. La définition de type de document doit apparaître avant le premier élément du document. Le nom qui suit le mot DOCTYPE dans la définition de type de document doit correspondre au nom de l'élément racine.

Exemple de DTD *telephone.dtd*:

```
<!ELEMENT telephone (#PCDATA)>
```

Déclaration ELEMENT(première approche

La syntaxe est la suivante :
- **élément feuille** : <!ELEMENT nom catégorie>

ou si un élément en contient d'autres
- **élément branche :** <!ELEMENT nom (enfant1, enfant2, ...)>

Le type d'élément `telephone` permet des données en caractères, une chaîne de caractères telle que `123456` est donc un contenu correct.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE telephone SYSTEM "telephone.dtd">
<telephone>123456</telephone>
```

`#PCDATA` signifie certes à peu près "autant de texte que vous voulez", mais il signifie aussi "aucun autre élément intérieur". Une application comme
`<telephone><gras>123456</gras></telephone>`

n'est donc pas permise d'après la définition de l'élément `telephone`.

Le `P` dans `PCDATA` signifie, que le contenu est analysé complètement par l'analyseur syntaxique XML.

DTD interne :

<?xml version="1.0"?>
<!DOCTYPE root_element [contenu de la dtd]>
<root_element>
    contenu xml
</root_element>

DTD externe :

<?xml version="1.0"?>
<!DOCTYPE root_element SYSTEM "chemin_fichier">
<root_element>
    contenu xml
</root_element>

et dans le fichier xml lui-même

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

```
<!DOCTYPE donnee SYSTEM "structure.dtd"> <!--DTD-->
```

<donnees>

...

</donnees>

Vous avez toujours besoin d'une DTD quand vous désirez des données XML qui ne soient pas seulement **conformes, mais qui soient également valides.** Car "valide" signifie: pouvant être validées au vu d'une DTD.

Instruction DOCTYPE

L'instruction qui lie le fichier XML à une DTD prédéfinie est :

```
<!DOCTYPE element-racine  SYSTEM "URL_de_la_DTD" >
```

ou

```
<!DOCTYPE element-racine  PUBLIC "nom" "URL_de_la_DTD" >
```

Dans le cas de XHTML 1.0 Strict, ce sera :

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

L'élément racine

est simplement le nom de la balise principale, pour XHTML, ce sera `html`.

Les mot-clef `SYSTEM` ou `PUBLIC`

soit `SYSTEM`, soit `PUBLIC` indique le moyen de retrouver la DTD.

1. Dans le premier cas, il s'agit d'une DTD que nous spécifions nous même, et que nous avons rendue disponible à l'adresse URL spécifiée.
2. Dans le second cas, il s'agit d'une DTD définie et **publiée par un organisme officiel**, le nom permet de l'identifier.
   Par exemple, XHTML est défini par le W3C et les navigateurs peuvent retrouver cette DTD par son nom (`-//W3C//DTD XHTML 1.0 Strict//EN`). Au cas où par exemple le navigateur ne disposerait pas de la DTD relative au nom, l'URL permet de retrouver la DTD de XHTML 1.0 Strict à l'URL :
   `http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd`
   C'est aussi le moyen de la récupérer dans son navigateur pour l'observer... Cette DTD compte un millier de lignes !

L'attribut `xmlns`

L'attribut `xmlns`, pour XML Naming Space, indique une URL unique qui identifie de manière unique les éléments de XHTML par rapport à d'autres éléments qui pourrait entrer en conflit. Cet attribut apparaît dans le marqueur d'ouverture de l'élément racine du document XML, ici la balise

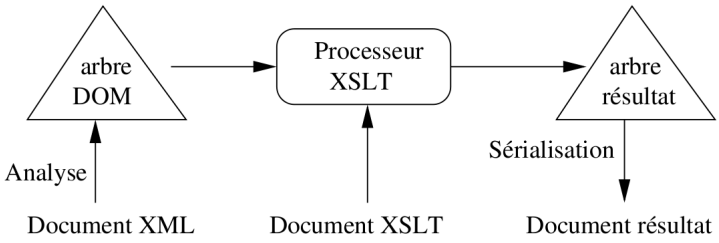
`<html>`.

l'attribut `xml:lang`

Enfin, l'attribut `xml:lang` permet de spécifier la langue relative à l'élément XML concerné. Le code est composé de deux lettres minuscules qui se rapportent à un code langue, d'un tiret et de deux lettres majuscules correspondant à un code pays.

Le détail des DTD sera vu en fin de cours car c'est un chapitre très technique et moins utile pour un usage simple du xml

XSLT



XSLT est, comme son nom l'indique, un langage destiné à transformer un fichier XML en quelque chose d'autre.
XSLT offre de nombreuses fonctions dignes d'un langage de haut-niveau : variables, paramètres, tests, boucles, fonctions, inclusion d'une feuille de style XSLT dans une autre, chargement de plusieurs documents XML dans une même feuille de style XSLT, recherche de balises XML selon de nombreux critères, etc.

Un site pratique:

http://wwww.laltruiste.com/document.php?url=http://wwww.laltruiste.com/coursxsl/sommaire.html

XSLT Element Reference

Structure d'une feuille XSL

xsl:stylesheet

élément racine de la feuille de style

xsl:output

permet de définir le type de sortie qui sera produit et de générer des entêtes.

<xsl:output
method = "xml" | "html" | "text"
version = nmtoken
encoding = string
omit-xml-declaration = "yes" | "no"
standalone = "yes" | "no"
doctype-public = string
doctype-system = string
indent = "yes" | "no"
media-type = string />

method="xml|html|text" *indique le format du document résultant.*

- Pour spécifier **une sortie HTML :**
  <xsl:output method="html" />

```
<xsl:output
   method="html"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"
```

```
   doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
   indent="yes"
   encoding="ISO-8859-1"
/>
```

- Pour spécifier **une sortie TEXT :**
  <xsl:output method="text" />

A mettre au début du fichier (après xsl:stylesheet)

<xsl:template> règle modèle

L'élément <xsl:template> permet de définir une règle de modèle.

Quelle est la syntaxe d'une règle modèle ?

Elle est la suivante :

<xsl:template match="*motif*"> *<!-- Spécifie à quels noeuds la règle est applicable -->*

<!-- Insertions dans le fichier cible de données prélevées/calculées à partir des données du fichier source. Appel éventuel d'autres règles modèles -->

</xsl:template>

Si vous souhaitez que le modèle fasse quelque chose, ne serait-ce que rendre la main à d'autres modèles, eh bien il faut le lui demander explicitement !

Vous pouvez, par exemple lui demander de simplement remplacer les balises <para> et tout ce qu'elles contiennent par le texte "trouvé !". En ce cas il vous suffira d'écrire :

```
<xsl:template match="//para">
trouvé !
</xsl:template>
```

Les attributs :

Attribut	Description
Match= "pattern"	traite les noeuds sélectionnés par le pattern spécifié, au lieu de traiter tous les éléments.
name="nom"	traite les noeuds sélectionnés par le pattern spécifié, au lieu de traiter tous les éléments.
Priority= "niveau"	applique les templates avec un mode donné.
mode="mode"	applique les templates avec un mode donné.

Les principaux motifs (patterns) sont :

Pattern	Exemple	Signification
|	**Gauche|Milieu**	Indique une alternative (un noeud ou bien l'autre (ou les deux))
/	**personne/nom**	Chemin d'accès aux éléments (*personne/bras/gauche*) au même titre que l'arborescence utilisée généralement pour les fichiers (*/usr/bin/toto*)
*	*	Motif "joker" désignant n'importe quel élément
//	**//personne**	Indique tous les descendants d'un noeud

Pattern	Exemple	Signification
.	**.**	Caractérise le noeud courant
..	**..**	Désigne le noeud parent
@	**@valeur**	Indique un attribut caractéristique

Exemple :

// Indique tous les descendants d'un noeud

Si vous désirez que ce texte "trouvé !" apparaisse à l'intérieur d'une nouvelle balise, par exemple <p>, alors vous écrirez :

```
<xsl:template match="//para">
<p>trouvé !</p>
</xsl:template>
```

La transformation peut être réalisée

- soit par ajout de texte,
- soit en définissant des *éléments de transformation*, c'est-à-dire des éléments permettant de définir des règles de transformation à appliquer aux éléments sélectionnés par l'attribut *match*

L'attribut *select* permet de spécifier certains éléments enfants auxquels la transformation doit être appliquée

La première chose à faire quand on commence une feuille XSL devant être appliquée à un fichier XML, c'est d'écrire

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

pour se placer à la racine du document.

L'élément <xsl:template> accepte en son sein les éléments suivants :

- xsl:apply-templates
- xsl:attribute
- xsl:call-template
- xsl:choose
- xsl:comment
- xsl:copy
- xsl:copy-of
- xsl:element
- xsl:for-each
- xsl:if
- xsl:param
- xsl:processing-instruction
- xsl:text
- xsl:value-of
- xsl:variable
- éléments HTML

<xsl:apply-templates select= ".." />

Il suffira ensuite d'utiliser l'élément <xsl:apply-templates> afin d'effectuer la mise en forme des éléments ciblés.

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

L'élément apply-templates utilisé au sein de la balise <xsl:template match="..."> permet d'appliquer la règle de transformation contenu dans la balise template.

L'élément **<xsl:apply-templates/>** indique le traitement de tous les enfants directs de la racine.

les attributs suivants:

select= (facultatif) choisit un modèle qui doit être appliqué. Le jeu de nœuds désiré est mentionné ou bien un chemin d'après la syntaxe XPath.
Quand une définition xsl:template existe pour cet élément, elle est appliquée.
Si l'attribut select manque, toutes les définitions xsl:template de l'échelon immédiatement inférieur sont appliquées.

mode= (facultatif) Ne choisit le modèle mentionné avec select= que lorsqu'il a le mode mentionné. Pour cela il faut que lors de la définition de xsl:template un nom qui concorde ait été attribué pour le mode avec mode=.

Peut être placé à l'intérieur de xsl:template.

<xsl:value-of>

Definition et Usage

L'élément *<xsl:value-of>* permet d'extraire la valeur d'un noeud sélectionné dans l'arborescence d'un document XML.

```
<xsl:value-of
                    select="expression"
                    disable-output-escaping="yes|no"/>
```

Cet élément ne peut traiter qu'un seul noeud à la fois contrairement à <xsl:apply-tempates> qui gére un ensemble de noeuds sélectionné par son *pattern*.

Utilisé en conjonction avec l'instruction <xsl:for-each>, l'élément *<xsl:value-of>* peut alors traiter une série de noeuds correspondant à l'expression de l'attribut *select*.

Les attributs :

Attribut	Description
select="pattern"	sélectionne des noeuds dans une arborescence source.
disable-output-escaping="yes|no"	active ou désactive le remplacement des caractères spéciaux par leur entité XML.

L'élément *<xsl:value-of>* peut non-seulement **sélectionner des éléments XML**, mais aussi **des attributs** par l'intermédiaire du signe @ ainsi que **des paramètres ou des variables** avec *$*.

L'élément précitée ne peut être contenu que dans une règle de modèle <xsl:template> ou un appel à cette dernière <xsl:call-template>.

Un exemple simple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="auteurs.xsl"?>
<Cours>
      <Titre>Cours XML</Titre>
      <Auteur>
            <Nom>Poulard</Nom>
            <Prenom>Philippe</Prenom>
      </Auteur>
      <Description>
      Ce cours aborde les concepts de base mis en &#339;uvre dans
XML.
      </Description>
</Cours>
```

Fichier auteurs.xsl

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/1999/xhtml">
<xsl:output method="html" encoding="ISO-8859-1"/>
<xsl:template match="/">
<html>
<head>
<title><xsl:value-of select="Cours/Titre"/></title>
</head>
<body>
<h1><xsl:value-of select="Cours/Titre"/></h1>
<xsl:apply-templates select="Cours/Auteur"/>
<xsl:apply-templates select="Cours/Description"/>
</body>
</html>
</xsl:template>
<xsl:template match="Auteur">
<p align="right">par <xsl:value-of select="Nom"/>
 <xsl:value-of select="Prénom"/></p>
</xsl:template>
<xsl:template match="Description">
<p><xsl:value-of select="."/></p>
</xsl:template>
</xsl:stylesheet>
```

dans le contexte de "/"

dans le contexte de "Auteur"

Exemple 2:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="stylesheet.xsl" ?>
<test>
1234567890
</test>
```

Fichier stylesheet.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
   method="html"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
   doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
   indent="yes"
   encoding="iso-8859-1"
/>
<xsl:template match="/">
 <html><head></head>
 <body style="background-color:#000000; font-family:Algerian; font-size:80px;
color:#33FF33">
  <xsl:value-of select="." />
 </body></html>
</xsl:template>

</xsl:stylesheet>
```

Explication:

Un processeur XSL analyse un fichier XML source et s'efforce de trouver une règle modèle concordante. S'il en trouve une, les instructions qu'elle contient sont évaluées.

Le traitement commence toujours par le modèle sélectionnant le noeud correspondant au motif match="/". Ce noeud est le noeud racine.

Les données XML de l'exemple ne contiennent que l'élément racine avec une suite de chiffres comme contenu. La feuille de style de l'exemple n'en est que plus simple.

Avec **xsl:template** est définie un modèle pour transcrire les données XML en sortie HTML. Ces définitions XSL et d'autres sont incluses dans l'élément racine `xsl:stylesheet`. Dans l'exemple le repère d'ouverture de cet élément contient les mentions typiques sur la version XSLT, sur la source XSLT et sur la source de l'espace de nommage HTML employé (ici: XHTML).

Lorsque ce modèle n'est pas explicitement indiqué, le modèle implicite est utilisé (il comporte une seule instruction). Cette instruction signifie : traiter tous les noeuds fils du noeud courant, y compris les noeuds textuels

Autre exemple

Considérons une liste de livres avec nom du livre, auteur et date

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="bdd.xsl" ?>
<bibliotheque>

      <livre>
            <nom>Le guet des orfèvres</nom>
            <auteur>Terry Pratchett</auteur>
            <date>1993</date>
      </livre>

      <livre>
            <nom>Nobliaux et sorcières</nom>
            <auteur>Terry Pratchett</auteur>
            <date>1992</date>
      </livre>

      <livre>
            <nom>Mécomptes de fées</nom>
            <auteur>Terry Pratchett</auteur>
            <date>1991</date>
      </livre>
      <livre>
            <nom>Tinbin obéit aux serviettes</nom>
            <auteur>Alain Provistt</auteur>
            <date>1999</date>
      </livre>

</bibliotheque>
```

et sa feuille xsl: bdd.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
   method="html"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
   doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
   indent="yes"
   encoding="ISO-8859-1"
/>

      <xsl:template match="/">
             <html>
             <head>
             </head>
             <body>
              <xsl:apply-templates />
             </body>
             </html>
      </xsl:template>

      <xsl:template match="/livre">

              <xsl:apply-templates />
      </xsl:template>

      <xsl:template match="nom">
            <h2>
              <xsl:value-of select="." />
            </h2>
      </xsl:template>

      <xsl:template match="auteur">
            <span style="color:blue">
              <xsl:value-of select="." />
            </span><br/>
      </xsl:template>

</xsl:stylesheet>
```

On remarque que la fin de chaque fiche est affichée sans demande précise;Lorsque le modèle n'est pas explicitement indiqué, le modèle implicite est utilisé (il comporte une seule instruction). Cette instruction signifie : traiter tous les noeuds fils du noeud courant, y compris les noeuds textuels.

Résultat

Le guet des orfèvres
Terry Pratchett
1993
Nobliaux et sorcières
Terry Pratchett
1992
Mécomptes de fées
Terry Pratchett
1991
Tinbin obéit aux serviettes
Alain Provistt
1999

Utiliser un attribut @

select="item/@date" permet de récupérer l'attribut date de item

Les attributs sont accessibles de la même manière que les éléments. Notez la présence du caractère **"@" devant les noms d'attributs**.

Source XML

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl"
href="style.xsl" ?>

<source>

<AAA id="a1" pos="start">
   <BBB id="b1"/>
   <BBB id="b2"/>
</AAA>
<AAA id="a2">
   <BBB id="b3"/>
   <BBB id="b4"/>
   <CCC id="c1">
      <DDD id="d1"/>
   </CCC>
   <BBB id="b5">
      <CCC id="c2"/>
   </BBB>
</AAA>

</source>
```

Feuille de style XSLT

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

<xsl:template match="AAA">
   <div style="color:purple">
      <xsl:value-of select="name()"/>
      <xsl:text> id=</xsl:text>
      <xsl:value-of select="@id"/>
   </div>
</xsl:template>


</xsl:stylesheet>
```

Sortie

```
<div style="color:purple">AAA
id=a1</div>

<div style="color:purple">AAA
id=a2</div>
```

Vue HTML

AAA id=a1
AAA id=a2

xsl:for-each

L'instruction xsl:for-each comporte un modèle qui est appliqué à chaque noeud sélectionné à l'aide de l'attribut select.
xsl:for-each n'est pas vraiment une boucle comme un "for" en langage C. Cette fonction va prendre tous les noeuds d'une requête XPATH, et va leur appliquer un traitement.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<source>

<AAA id="a1" pos="start">
     <BBB id="b1"/>
     <BBB id="b2"/>
</AAA>
<AAA id="a2">
     <BBB id="b3"/>
     <BBB id="b4"/>
     <CCC id="c1">
          <DDD id="d1"/>
     </CCC>
     <BBB id="b5">
          <CCC id="c2"/>
     </BBB>
</AAA>

</source>
```

BBB id=b1
BBB id=b2
BBB id=b3
BBB id=b4
BBB id=b5
CCC id=c1

Feuille de style XSLT

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version = '1.0'

     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="/">
     <xsl:for-each select="//BBB">
          <DIV style="color:red">
               <xsl:value-of select="name()"/>
               <xsl:text> id=</xsl:text>
               <xsl:value-of select="@id"/>
          </DIV>
     </xsl:for-each>
     <xsl:for-each select="source/AAA/CCC">
          <DIV style="color:navy">
               <xsl:value-of select="name()"/>
               <xsl:text> id=</xsl:text>
               <xsl:value-of select="@id"/>
          </DIV>
     </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Feuille de style dans le fichier xsl

Pour inclure la feuille de style directement dans le fichier xsl, il suffit d'utiliser <![CDATA[...]]>:

```
<style type="text/css">
    <![CDATA[
    h1 {....}
    ]]>
</style>
```

Exemple:liste.xsl et liste.xml

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

<xsl:output method="html" doctype-
system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd" doctype-public="-//W3C//DTD XHTML
1.0 Transitional//EN"   indent="yes" encoding="iso-
8859-1" />

<xsl:template match="/">
 <html>
 <head>
  <style type="text/css">
   <![CDATA[
  h1 {color:red}
  p {font-family:Arial,Helvetica,sans-serif;
font-size:12pt}
  b  {color:blue}
  ]]>
   </style>
 </head>
 <body>
  <xsl:apply-templates />
 </body>
 </html>
</xsl:template>

<xsl:template match="titre">
 <h1><xsl:value-of select="." /></h1>
</xsl:template>

<xsl:template match="glossaire/element">
 <p>
  <xsl:apply-templates />
 </p>
</xsl:template>

<xsl:template match="terme">
 <b ><xsl:apply-templates />: </b>
</xsl:template>

<xsl:template match="signification">
  <xsl:value-of select="." />
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="ISO-8859-
1"?>
<?xml-stylesheet type="text/xsl"
href="liste.xsl" ?>
<test>
 <titre>Extensions de fichiers</titre>
 <glossaire>
  <element>
   <terme>bak</terme>
   <signification>
    fichier de
    sauvegarde
    </signification>
  </element>
  <element>
   <terme>bmp</terme>
   <signification>
    graphique Bitmap
    </signification>
  </element>
 </glossaire>
</test>
```

Élément <xsl:text>

L'élément <xsl:text> est utilisé pour écrire du texte en sortie.

TEST

- **l'égalité** : =
- **supérieur** : &gt ;
- **inférieur** : &lt ;
- **addition** : +
- **soustraction** : - (attention ce symbole est accepté dans les noms de balises, il faut donc impérativement le faire précéder et succéder d'un espace si on veut qu'il soit interprété)
- **division** : div
- **modulo** : mod

xsl:if

L'élément *<xsl:if>* permet d'appliquer un test conditionnel dans la structure d'une feuille de style XSL.

```
<xsl:if test="condition">
   Instructions...
</xsl:if>
```

Si le test conditionnel est vérifié alors le processeur XSL exécutera les instructions contenues à l'intérieur des marqueurs, sinon il les ignorera et passera aux instructions suivantes.

Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="invites.xsl" ?>
<liste>
   <invite>Moi</invite>
   <invite>Jean</invite>
   <invite>Michel</invite>
</liste>
```

invites.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
   method="html"
   encoding="ISO-8859-1"
/>
 <xsl:template match="/">
 <ul>
   <xsl:for-each select="//invite">
      <li>
         <xsl:apply-templates select="." />
      <xsl:if test=".='Michel'">
            <i>
             <xsl:text> bonjour Michel </xsl:text>
            </i>
             </xsl:if>
      </li>
```

```
   </xsl:for-each>
</ul>
</xsl:template>
</xsl:stylesheet>
```

Si le test conditionnel est vérifié alors le processeur XSL exécutera les instructions contenues à l'intérieur des marqueurs, sinon il les ignorera et passera aux instructions suivantes.

xsl:choose - xsl:otherwise

La fonction **xsl:choose** permet d'exécuter différents codes selon différentes conditions. Exemple d'utilisation de la l'instruction choose :

On utilise souvent xsl:choose comme alternative au xsl:if pour disposer de **xsl:otherwise** (qui remplace xsl:else qui n'existe pas).

On utilise souvent xsl:choose comme alternative au xsl:if pour disposer de xsl:otherwise (qui remplace xsl:else qui n'existe pas).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="invites.xsl" ?>
<liste>
   <invite>Moi</invite>
   <invite>Jean</invite>
   <invite>Michel</invite>
</liste>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
   method="html"
   encoding="ISO-8859-1"
/>
 <xsl:template match="/">
 <ul>
   <xsl:for-each select="//invite">
        <li>
            <xsl:apply-templates select="." />
   <xsl:choose>
            <xsl:when test=".= 'Moi'">
            <xsl:text> Salut</xsl:text>
            </xsl:when>

            <xsl:when test=".= 'Michel'">
            <xsl:text> Fait ciseau</xsl:text>
            </xsl:when>

            <xsl:otherwise>
                <xsl:text> Message impersonnel Bonjour</xsl:text>
            </xsl:otherwise>

   </xsl:choose>
        </li>
   </xsl:for-each>
</ul>
</xsl:template>
</xsl:stylesheet>
```

For-each: autre exemple -catalogue cd-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<catalog>
	<cd>
		<title>Empire Burlesque</title>
		<artist>Bob Dylan</artist>
		<country>USA</country>
		<company>Columbia</company>
		<price>10.90</price>
		<year>1985</year>
	</cd>
	<cd>
		<title>Hide your heart</title>
		<artist>Bonnie Tyler</artist>
		<country>UK</country>
		<company>CBS Records</company>
		<price>9.90</price>
		<year>1988</year>
	</cd>
	<cd>
		<title>Greatest Hits</title>
		<artist>Dolly Parton</artist>
		<country>USA</country>
		<company>RCA</company>
		<price>9.90</price>
		<year>1982</year>
	</cd>
	<cd>
		<title>Still got the blues</title>
		<artist>Gary Moore</artist>
		<country>UK</country>
		<company>Virgin records</company>
		<price>10.20</price>
		<year>1990</year>
	</cd>
	<cd>
		<title>Eros</title>
		<artist>Eros Ramazzotti</artist>
		<country>EU</country>
		<company>BMG</company>
		<price>9.90</price>
		<year>1997</year>
	</cd>
	<cd>
		<title>One night only</title>
		<artist>Bee Gees</artist>
		<country>UK</country>
		<company>Polydor</company>
		<price>10.90</price>
		<year>1998</year>
	</cd>
	<cd>
		<title>Sylvias Mother</title>
		<artist>Dr.Hook</artist>
		<country>UK</country>
		<company>CBS</company>
		<price>8.10</price>
		<year>1973</year>
	</cd>
	<cd>
```

```
      <title>Maggie May</title>
      <artist>Rod Stewart</artist>
      <country>UK</country>
      <company>Pickwick</company>
      <price>8.50</price>
      <year>1990</year>
</cd>
<cd>
      <title>Romanza</title>
      <artist>Andrea Bocelli</artist>
      <country>EU</country>
      <company>Polydor</company>
      <price>10.80</price>
      <year>1996</year>
</cd>
<cd>
      <title>When a man loves a woman</title>
      <artist>Percy Sledge</artist>
      <country>USA</country>
      <company>Atlantic</company>
      <price>8.70</price>
      <year>1987</year>
</cd>
<cd>
      <title>Black angel</title>
      <artist>Savage Rose</artist>
      <country>EU</country>
      <company>Mega</company>
      <price>10.90</price>
      <year>1995</year>
</cd>
<cd>
      <title>1999 Grammy Nominees</title>
      <artist>Many</artist>
      <country>USA</country>
      <company>Grammy</company>
      <price>10.20</price>
      <year>1999</year>
</cd>
<cd>
      <title>For the good times</title>
      <artist>Kenny Rogers</artist>
      <country>UK</country>
      <company>Mucik Master</company>
      <price>8.70</price>
      <year>1995</year>
</cd>
<cd>
      <title>Big Willie style</title>
      <artist>Will Smith</artist>
      <country>USA</country>
      <company>Columbia</company>
      <price>9.90</price>
      <year>1997</year>
</cd>
<cd>
      <title>Tupelo Honey</title>
      <artist>Van Morrison</artist>
      <country>UK</country>
      <company>Polydor</company>
      <price>8.20</price>
      <year>1971</year>
</cd>
```

```
<cd>
      <title>Soulsville</title>
      <artist>Jorn Hoel</artist>
      <country>Norway</country>
      <company>WEA</company>
      <price>7.90</price>
      <year>1996</year>
</cd>
<cd>
      <title>The very best of</title>
      <artist>Cat Stevens</artist>
      <country>UK</country>
      <company>Island</company>
      <price>8.90</price>
      <year>1990</year>
</cd>
<cd>
      <title>Stop</title>
      <artist>Sam Brown</artist>
      <country>UK</country>
      <company>A and M</company>
      <price>8.90</price>
      <year>1988</year>
</cd>
<cd>
      <title>Bridge of Spies</title>
      <artist>T`Pau</artist>
      <country>UK</country>
      <company>Siren</company>
      <price>7.90</price>
      <year>1987</year>
</cd>
<cd>
      <title>Private Dancer</title>
      <artist>Tina Turner</artist>
      <country>UK</country>
      <company>Capitol</company>
      <price>8.90</price>
      <year>1983</year>
</cd>
<cd>
      <title>Midt om natten</title>
      <artist>Kim Larsen</artist>
      <country>EU</country>
      <company>Medley</company>
      <price>7.80</price>
      <year>1983</year>
</cd>
<cd>
      <title>Pavarotti Gala Concert</title>
      <artist>Luciano Pavarotti</artist>
      <country>UK</country>
      <company>DECCA</company>
      <price>9.90</price>
      <year>1991</year>
</cd>
<cd>
      <title>The dock of the bay</title>
      <artist>Otis Redding</artist>
      <country>USA</country>
      <company>Atlantic</company>
      <price>7.90</price>
      <year>1987</year>
```

```
	</cd>
	<cd>
		<title>Picture book</title>
		<artist>Simply Red</artist>
		<country>EU</country>
		<company>Elektra</company>
		<price>7.20</price>
		<year>1985</year>
	</cd>
	<cd>
		<title>Red</title>
		<artist>The Communards</artist>
		<country>UK</country>
		<company>London</company>
		<price>7.80</price>
		<year>1987</year>
	</cd>
	<cd>
		<title>Unchain my heart</title>
		<artist>Joe Cocker</artist>
		<country>USA</country>
		<company>EMI</company>
		<price>8.20</price>
		<year>1987</year>
	</cd>
</catalog>
```

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:output
   method="html"
   encoding="ISO-8859-1"
/>
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each
     select="catalog/cd">
     <xsl:if test="price>10">
           <tr>
           <td><xsl:value-of
           select="title"/></td>
           <td><xsl:value-of
           select="artist"/></td>
           </tr>
     </xsl:if>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Pour chaque cd du catalogue

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore
One night only	Bee Gees
Romanza	Andrea Bocelli
Black angel	Savage Rose
1999 Grammy Nominees	Many

xsl:sort

Tri alphabétique **"ascending"** par défaut

<xsl:sort select="..." order="descending"/>

Tri numérique

<xsl:sort select="..." data-type="number" order="descending"/>

Trier les éléments d'après leur contenu.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="sort.xsl" ?>
<test>
<joueur><nom>Antoine</nom><points>12</points></joueur>
<joueur><nom>Patrick</nom><points>19</points></joueur>
<joueur><nom>Richard</nom><points>27</points></joueur>
<joueur><nom>William</nom><points>10</points></joueur>
</test>
```

fichier sort.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output
   method="html"
   encoding="ISO-8859-1"
/>
<xsl:template match="/">
 <html>
 <head>
 </head>
 <body>
 <table border="1">
  <xsl:for-each select="test/joueur">
   <xsl:sort select="points" order="descending" data-type="number" />
   <tr>
   <td><xsl:value-of select="nom" /></td>
   <td><xsl:value-of select="points" /></td>
   </tr>
  </xsl:for-each>
 </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Élément <xsl:number>

http://msdn.microsoft.com/fr-fr/library/ms256084%28v=vs.80%29.aspx

Insère un nombre formaté dans l'arborescence résultat.

```
<xsl:number
  level = "single" | "multiple" | "any"
  count = Pattern
  from = Pattern
  value = number-expression
  format = { string }
  lang = { nmtoken }
  letter-value = { "alphabetic" | "traditional" }
  grouping-separator = { char }
  grouping-size = { number } />
```

Attributs

---

level

Spécifie les niveaux de l'arborescence source à prendre en considération ; les valeurs possibles sont `"single"`, `"multiple"` et `"any"`. La valeur par défaut est `"single"`.

count

Modèles qui spécifie les nœuds à compter à ces niveaux. Si l'attribut `count` n'est pas spécifié, la valeur par défaut est le modèle correspondant à tout nœud ayant le même type de nœud que le nœud actuel et, si le nœud actuel a un nom développé, ayant le même nom développé que le nœud actuel.

from

Modèles qui spécifie où commence le comptage.

value

Spécifie l'expression à convertir en un nombre et à insérer dans l'arborescence résultat. Si aucun attribut `value` n'est spécifié, l'élément `<xsl:number>` insère un nombre d'après la position du nœud actuel dans l'arborescence source.

format

Séquence de jetons spécifiant le format à utiliser pour chaque nombre figurant dans la liste. S'il n'y a pas de jetons de format, la valeur par défaut utilisée est 1, qui génère une séquence 1 2 ... 10 11 12.... Chaque nombre hormis le premier est séparé du précédent par le jeton séparateur, lequel précède le jeton de format utilisé pour formater ce nombre. S'il n'y a pas de jetons séparateurs, c'est un point qui est utilisé (« . »).

.

Jeton de format	Séquence générée
1	1 2 3 4 5 ... 10 11 12 ...
01	01 02 03 ... 19 10 11 ... 99 100 101...
A	A B C ... Z AA AB AC...
i	i ii iii iv v vi vii viii ix x...
I	I II III IV V VI VII VIII IX X...

lang

Spécifie l'alphabet utilisé. Si aucune langue n'est spécifiée, elle est déterminée par l'environnement système.

letter-value

Lève l'ambiguïté entre les séquences de numérotation qui utilisent des lettres. Une séquence de numérotation affecte des valeurs numériques aux lettres dans l'ordre alphabétique, tandis que l'autre affecte des valeurs numériques à chaque lettre d'une autre manière traditionnelle pour cette langue. En anglais, ces séquences de numérotation sont spécifiées par les jetons de format « a » et « i ». Dans certaines langues, le premier membre de chaque séquence est identique, si bien que le jeton de format seul serait ambigu. La valeur `"alphabetic"` spécifie l'ordre alphabétique ; la valeur `"traditional"` spécifie l'autre possibilité. La valeur par défaut est `"alphabetic"`.

grouping-separator

Définit le séparateur utilisé pour le regroupement (p. ex. des milliers) dans les séquences de numérotation décimales. Par exemple, `grouping-separator=","` et `grouping-size="3"` produiraient des nombres du genre 1,000,000. Si un seul des attributs `grouping-separator` et `grouping-size` est spécifié, il est ignoré.

grouping-size

Spécifie la taille (normalement 3) des groupes. Par exemple, `grouping-separator=","` et `grouping-size="3"` produiraient des nombres du genre 1,000,000. Si un seul des attributs `grouping-separator` et `grouping-size` est spécifié, il est ignoré.

Les langues/schémas de numérotation suivants sont pris en charge. « Jeton de format » correspond à l'attribut `format`, « Langue » correspond à l'attribut `lang` et « Valeur lettre » correspond à l'attribut `letter-value`.

Description	Jeton de format	Langue	Valeur lettre
Occidental	0x0031 (1)	n/a	n/a
Lettre majuscule	0x0041 (A)	n/a	n/a
Lettre minuscule	0x0061 (a)	n/a	n/a
Chiffre romain majuscule	0x0049 (I)	n/a	n/a
Chiffre romain minuscule	0x0069 (i)	n/a	n/a
Russe (cyrillique) majuscule	0x0410	n/a	n/a
Russe (cyrillique) minuscule	0x0430	n/a	n/a
Hébreu alphabétique	0x05d0	n/a	Alphabétique
Hébreu traditionnel	0x05d0	n/a	Arabe
traditionnel	0x0623	n/a	n/a
Hindi, consonnes	0x0905	n/a	n/a
Hindi, voyelles	0x0915	n/a	n/a
Hindi, chiffres	0x0967	n/a	n/a
Thaï, lettres	0x0e01	n/a	n/a
Thaï, chiffres	0x0e51	n/a	n/a
Japonais, Aiueo (double octet)	0x30a2	n/a	n/a
Japonais, Iroha (double octet)	0x30a4	n/a	n/a

Description	Jeton de format	Langue	Valeur lettre
Coréen, Chosung	0x3131	n/a	n/a
Taïwanais décimal	0x4e01	« zh-tw »	n/a
Coréen décimal	0x4e01	« ko »	n/a
Asiatique décimal	0x4e01	toute autre langue	n/a
Asiatique, Kanji	0x58f1	n/a	n/a
Taïwanais traditionnel	0x58f9	« zh-tw »	n/a
Chinois traditionnel	0x58f9	toute autre langue	n/a
Chinois « Zodiaque » 12	0x5b50	n/a	n/a
Chinois « Zodiaque » 10	0x7532	n/a	n/a
Chinois « Zodiaque » 60	0x7532, 0x5b50	n/a	n/a
Coréen, Ganada	0xac00	n/a	n/a
Coréen décimal	0xc77c	n/a	n/a
Coréen 99	0xd558	n/a	n/a
Occidental (double octet)	0xff11	n/a	n/a
Japonais, Aiueo (simple octet)	0xff71	n/a	n/a
Japonais, Iroha (simple octet)	0xff72	n/a	n/a

Si le jeton de format à lui seul suffit pour lever l'ambiguïté pour un schéma de numérotation

particulier, il n'est pas nécessaire de spécifier la langue ni la valeur lettre.

Exemple :

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="numelem.xsl" ?>
<items>
   <item>voiture</item>
   <item>Plume</item>
   <item>LP 33 tours</item>
   <item>Sagesse</item>
   <item>GSM</item>
   <item>Projecteur</item>
   <item>trou</item>
   <item>Verrière</item>
   <item>Widget</item>
   <item>Concepte</item>
   <item>Null character</item>
</items>
```

et son fichier xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="html"/>

 <xsl:template match="items">
    <xsl:text>nombre d'items :</xsl:text>
    <xsl:value-of select="count(item)"/> <br/>
   <xsl:for-each select="item">
      <xsl:sort select="."/>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="."/>
      <blockquote>
      <xsl:number value="position()" format="&#x0069;-> "/>
      <xsl:value-of select="."/>
      </blockquote>
    <br/>
   </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

template et call-template

Il existe deux versions de l'élément template qui se différencient par leurs attributs :

<xsl:template match="XPath" > ... </xsl:template>	<xsl:template name="nom_modele"> ... </xsl:template>

La version avec l'attribut **match** définit un modèle par défaut, applicable à tel type d'élément. Ce modèle est instancié par le processeur XSLT à chaque fois qu'un élément correspondant est rencontré.
La version avec l'attribut **name** est destinés à êtres "appelés" par l'instruction **call-template.**
Exemples
XSL

```
<xsl:template match="/" >
    <xsl:call-template name="dis_bonjour"/>
</xsl:template>
<xsl:template name="dis_bonjour">
    Bonjour !
</xsl:template>
```

Quelque soit le document XML en entrée, le processeur commence à chercher un modèle pouvant s'appliquer à la racine, trouve ici le premier modèle dans la feuille de style et l'applique. Ce modele spécifie la génération de texte et l'instanciation du modèle dont le nom est *dis_bonjour*.
Il est possible, comme pour une fonction, de passer à un tel modèle des paramètres et de réaliser des appels récursifs :

```
<xsl:template match="/" >
    énumération :
    <xsl:call-template name="enumer">
        <xsl:with-param name="start">0</xsl:with-param>
        <xsl:with-param name="end">9</xsl:with-param>
    </xsl:call-template>
</xsl:template>

<xsl:template name="enumer">
    <xsl:param name="start"/>
    <xsl:param name="end"/>

    <xsl:value-of select="$start"/>

    <xsl:if test="$start &lt; $end">
        <xsl:call-template name="enumer">
            <xsl:with-param name="start"><xsl:value-of select="$start +
1"/></xsl:with-param>
            <xsl:with-param name="end"><xsl:value-of select="$end"/></xsl:with-
param>
        </xsl:call-template>
    </xsl:if>
</xsl:template>
Réponse
énumération :

0123456789
```

Explications des termes d'une DTD

Il y a quatre mots-clés dans les déclarations de : ELEMENT, ATTLIST, ENTITY et NOTATION.

Déclaration ELEMENT

La syntaxe est la suivante :
- élément **feuille** : <!ELEMENT nom catégorie>
- élément **branche** : <!ELEMENT nom (enfant1, enfant2, ...)>

Il y a trois catégories :

Type prédéfini	Description
ANY	L'élément peut contenir tout type de données
EMPTY	L'élément ne contient pas de données spécifiques (élément vide qui s'écrira <nom/>)
#PCDATA	L'élément doit contenir une chaîne de caractères

Ainsi un élément nommé *Nom* contenant un type #PCDATA sera déclaré de la façon suivante dans la DTD :

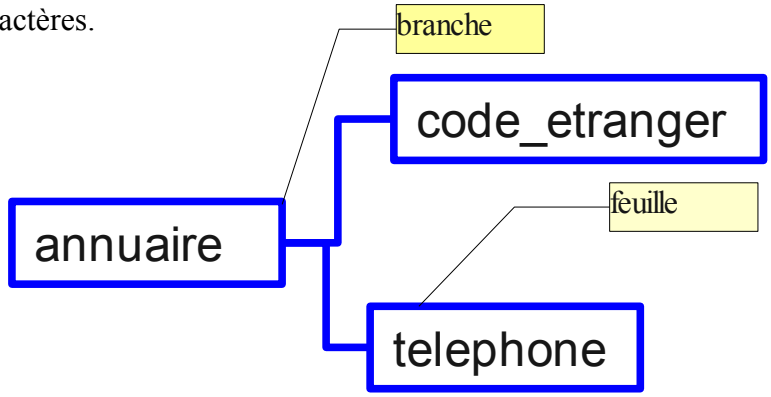
```
<! ELEMENT Nom (#PCDATA) >
```

Exemple de DTD *annuaire.dtd*:

```
<!ELEMENT annuaire (code_etranger, telephone)>
<!ELEMENT code_etranger (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
```

Explication:

Dans l'exemple trois types d'éléments sont définis. La première des trois définitions est un type d'élément avec un contenu d'éléments. Cette définition a la même construction que la définition d'un type d'élément contenant des caractères.



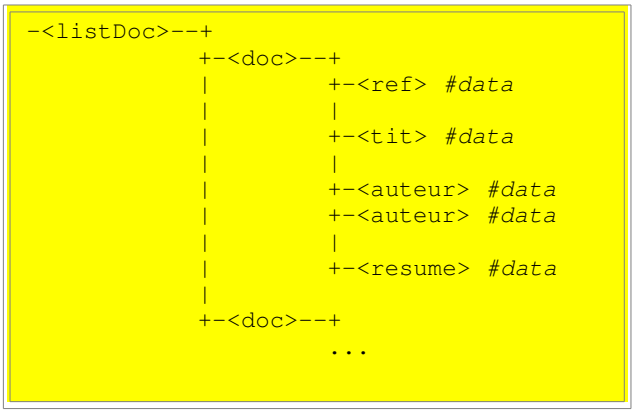
Exemple d'une application valide:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE annuaire SYSTEM "annuaire.dtd">
<annuaire>
  <code_etranger>0049</code_etranger>
  <telephone>974791003</telephone>
```

```
</annuaire>
```

Exemple 2 de DTD

Soit la structure suivante (correspondant à une liste de documents simples) :



Exemple de liste de document

```
<listDoc>
    <doc>
       <ref>B01</ref>
       <titre>bavardages</titre>
       <auteur>LaParlote</auteur>
       <resume>blabla inutile.</resume>
    </doc>
    <doc>
        ...
    </doc>
    ...
</listDoc>
```

Exemple de DTD correspondante

```
<!DOCTYPE listDoc [
<!ELEMENT listDoc (doc*)>
<!ELEMENT doc     (ref, tit, auteur*, resume)>
<!ELEMENT tit     (#PCDATA)>
<!ELEMENT auteur  (#PCDATA)>
<!ELEMENT resume  (#PCDATA)>
]>
```

Element non terminal (feuille)

```
<!ELEMENT identifieur de l'élément ( liste de citation d'éléments )>
```

Element terminal

```
<!ELEMENT identifieur de l'élément - - ( #PCDATA )>
```

Méta-caractères de répétition

D'autre part il est possible de définir des règles d'utilisation, c'est-à-dire les éléments XML qu'un élément peut ou doit contenir. Cette syntaxe se fait à l'aide de notations spécifiques dont voici un récapitulatif :

Opérateur	Signification	Exemple
+	L'élément doit être présent au minimum une fois	A+
*	L'élément peut être présent plusieurs fois (ou aucune)	A*
?	L'élément peut être optionnellement présent	A?
|	L'élément A ou l'élément B peuvent être présents	A|B
,	L'élément A doit être présent et suivi de l'élément B	A,B
()	Les parenthèses permettent de regrouper des éléments afin de leur appliquer les autres opérateurs	(A,B)+

Ainsi on peut créer la déclaration suivante dans la DTD :

```
<!ELEMENT personne (nom,prenom,telephone),email? >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT telephone (#PCDATA) >
<!ELEMENT email (#PCDATA) >
```

Cette déclaration pourra donc donner un document XML du style :

```
<personne>
        <nom>Pillou</nom>
        <prenom>Jean-Francois</prenom>
        <telephone>555-123456</telephone>
        <email>webmaster@commentcamarche.net</email>
</personne>
```

ou bien

```
<personne>
        <nom>Pillou</nom>
        <prenom>Jeff</prenom>
        <telephone>555-542136</telephone>
</personne>
```

Un élément peut avoir **un contenu mixte**, il s'écrit alors : <!ELEMENT nom (#PCDATA, enfant1, enfant2, ...)>, #PCDATA en premier

Un élément qui apparaît ainsi ne doit avoir qu'une seule instance obligatoire. Dans les autres cas, un opérateur de cardinalité est nécessaire :
? : une seule instance facultative : <!ELEMENT table (caption?,...
* : plusieurs instances facultatives;
+ : une ou plusieurs instances obligatoires : <!ELEMENT ul (li)+>.

Approfondissons :

Définir autant de répétitions que l'on veut pour un type d'élément

En l'absence de toute autre mention, un type d'élément ne peut être placé qu'une fois. Afin que l'élément d'un type d'élément puisse être noté plusieurs fois, vous devez l'exprimer dans la DTD.

Exemple de DTD recettes.dtd:

```
<!ELEMENT recettes (liste_ingredients, suite_instructions)>

<!ELEMENT liste_ingredients (ingredient)+>
<!ELEMENT ingredient (#PCDATA)>

<!ELEMENT suite_instructions (instruction)*>
<!ELEMENT instruction (#PCDATA)>
```

Explication:

Le type d'élément `liste_ingredients` peut contenir **un ou plusieurs types d'éléments** `ingredient`. La raison en est le **signe plus +** dans la définition derrière la mention du contenu. Par le signe plus, vous mentionnez que le contenu, dans l'exemple le type d'élément `ingredient`, doit être placé au moins une fois, autrement aussi souvent que souhaité dans `liste_ingredients`. Pour l'exemple de la recette, une telle construction est judicieuse car une recette doit comporter au moins un ingrédient.

Le type d'élément `suite_instructions` peut contenir de la même façon un ou plusieurs types d'élément `instruction` mais peut cependant également rester vide. La raison dans ce cas-ci est l'**étoile *** dans la définition derrière la mention du contenu. Par l'étoile, vous mentionnez que le contenu, dans l'exemple le type d'élément `instruction`, peut ne pas être placé, être placé une fois ou aussi souvent que souhaité dans `suite_instructions`. Pour l'exemple de la recette, cela signifie qu'il peut aussi y avoir des recettes avec ingrédients mais sans instructions.

Explication:

Les types d'éléments sont imbriqués de la façon définie dans la DTD. Les types d'éléments `ingredient` et `instruction` sont placés plusieurs fois dans `liste_ingredients` ou dans `suite_instructions`, ce qui est permis d'après la définition.

Attention:

Si l'exemple ne devait contenir aucun élément du type `instruction`, ce qui d'après la définition serait tout à fait permis grâce à l'étoile, vous devriez malgré tout noter dans l'explication valide: `<suite_instructions></suite_instructions>`
La raison en est que, lors de la définition du type d'élément `recettes` dans l'exemple ci-dessus, on a fixé qu'il devait comporter les deux types d'élément `liste_ingredients` et `suite_instructions`, et que ces deux types d'éléments doivent être placés, à savoir exactement une fois.

Définir des types d'éléments en alternative ou facultatifs pour le contenu d'éléments

Quand vous définissez des types d'élément avec un contenu d'éléments, il peut arriver que cette suite rigide d'éléments que vous prévoyez pour l'élément, ne soit pas toujours appropriée dans la pratique. Pour une adresse postale par exemple, on pourrait faire la distinction alternativement entre la mention de la rue et du numéro et une mention de boite postale, et la mention du titre d'une personne est un exemple typique de mention facultative, quelque chose donc que l'on désire mentionner pour quelques adresses mais par contre pas pour d'autres.

Exemple de DTD adresses.dtd:

```
<!ELEMENT adresses (adresse)*>

<!ELEMENT adresse (titre?, nom, (boite_postale | numero_rue), codepostal_ville)>

<!ELEMENT titre (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT boite_postale (#PCDATA)>
<!ELEMENT numero_rue (#PCDATA)>
<!ELEMENT codepostal_ville (#PCDATA)>
```

Le titre est défini dans l'exemple comme étant facultatif. Cela est permis par le **point d'interrogation** `?` placé dans la définition derrière la mention de contenu prés du nom du type d'élément. La simple notation de `titre` définirait l'utilisation du type d'élément comme étant absolument indispensable, alors que la mention `titre?` désigne son utilisation comme étant facultative.

Par ailleurs il a été défini dans l'exemple que le choix est possible entre une mention de boite postale ou d'adresse géographique. Pour cela, les types d'éléments au choix sont mis dans de nouvelles parenthèses. Dans ces parenthèses les types d'élément au choix sont notés, séparés par une **barre verticale** `|`.

Exemple d'une application valide:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE adresses SYSTEM "adresses.dtd">
<adresses>
<adresse>
  <nom>Société Générale</nom>
  <boite_postale>7001</boite_postale>
  <codepostal_ville>13100 Aix en Provence</codepostal_ville>
</adresse>
<adresse>
  <titre>Monsieur</titre>
  <nom>Luc Minighetti</nom>
  <numero_rue>112 rue de Lyon</numero_rue>
  <codepostal_ville>13000 Marseille</codepostal_ville>
</adresse>
</adresses>
```

Définir des éléments avec un contenu mixte

Quand vous désirez définir des applications XML dans lesquelles ne doivent pas être groupés de façon si rigide, mais peuvent être notés dans un ordre relativement libre, - comme par exemple en HTML - alors, vous devez définir des éléments avec un contenu mixte. En HTML `<body>...</body>` en est un tel élément typique. Dans cet élément, vous pouvez noter la plupart des autres éléments HTML dans un ordre relativement libre et en respectant quelques règles d'imbrication peu nombreuses. Pour les applications qui sont en général moins orientées sur les bases de données pour l'être davantage sur le texte libre, vous avez besoin de ces contenus mixtes.

Exemple de DTD texte.dtd:

```
<!ELEMENT texte (#PCDATA | menace | rire | question | cynique)*>
<!ELEMENT menace (#PCDATA)>
<!ELEMENT rire (#PCDATA | clignant_oeil)*>
<!ELEMENT question (#PCDATA)>
<!ELEMENT clignant_oeil (#PCDATA)>
<!ELEMENT cynique (#PCDATA)>
```

Exemple d'une application valide:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE text SYSTEM "texte.dtd">
<texte>
Quelquefois la conscience nous dit: <menace>tu dois penser
 davantage aux femmes et aux hommes.</menace>
Alors, bien sûr, on se demande parfois
, <question>pourquoi il y a là tant
matière à reflexion,</question>
mais quelquefois aussi, on obéit et réfléchit.
La femme dit à l'homme: <rire>Oh chéri, <clignant_oeil>tu vaux
ton poids d'or!</clignant_oeil></rire>
Et l'homme répond: <cynique>Oui, parceque je pousse le chariot
 et que j'y ai mis une pièce de deux €!</cynique>
La femme rétorque: <rire>tu as tout compris!</rire>
</texte>
```

Explication:

L'exemple d'application contient l'élément document `<texte>..</text>` qui contient un contenu mixte typique. Du texte normal est placé (données en caractères) mais également des autres éléments. L'élément `rire` est placé deux fois en tout. Cela pourrait tout aussi bien être l'élément `menace` qui serait utilisé deux ou plusieurs fois dans l'élément `texte`. Cela est permis en raison de l'utilisation de l'étoile `*` lors de la définition de tous les types d'élément intérieurs. L'élément `clignant_oeil` est placé à l'intérieur de `rire` comme c'est permis par la DTD - à l'extérieur il ne pourrait cependant pas être placé.

Définir des éléments au contenu libre

Les éléments au contenu libre sont une forme plus souple encore des éléments à contenu mixte. Il s'agit ici plus ou moins d'éléments joker ou passe-partout, dont le contenu n'est pas du tout fixé. Tous les autres types d'élément définis dans la DTD peuvent être placés dans un élément au contenu libre.

Exemple de DTD anytext.dtd:

```
<!ELEMENT anytext ANY>
<!ELEMENT anglais (#PCDATA)>
<!ELEMENT italiano (#PCDATA)>
```

Explication:

Pour toutes les définitions de l'exemple, le schéma de définition de types d'élément s'applique. Par le mot clé ANY (**en majuscules**) au lieu de la définition concrète du contenu d'un élément, vous mentionnez que cet élément (dans l'exemple l'élément document) peut contenir des contenus de caractères et des types d'élément avec un contenu d'éléments. Dans l'exemple, deux autres types d'éléments sont définis qui peuvent être eux aussi placés.

Exemple d'une application valide:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE anytext SYSTEM "anytext.dtd">
<anytext>
c'est un peu de  texte qui signifie en anglais:
<anglais>this is some text</anglais> et en italien:
<italiano>ciò è un certo testo</italiano>
</anytext>
```

Explication:

Dans l'exemple l'élément document `<anytext>...</anytext>` est défini. Des données en caractères et tous les autres éléments de votre choix pour lesquels un type d'élément existe dans la DTD peuvent y être placés. L'exemple montre un mélange simple.

Définir des éléments vides sans contenu

Les éléments vides sont ceux qui n'ont pas de contenu. En HTML classique par exemple `<br>` ou bien `<img>` font partie de ce genre d'éléments. En HTML on les appelle aussi éléments autonomes. Dans la philosophie de XML, à vrai dire, un contenu est prévu pour chaque élément. Quand vous voulez définir des types d'élément qui n'ont pas de contenu, vous devez le mentionner spécialement. Même la notation des éléments dans l'application indique qu'il s'agit d'un cas particulier.

Exemple de DTD lignes_texte.dtd:

```
<!ELEMENT lignes_texte (#PCDATA | nouvelle_ligne)*>
<!ELEMENT nouvelle_ligne EMPTY>
```

Explication:

Dans l'exemple, deux types d'élément sont définis: un type d'élément lignes_texte pour l'élément document avec un contenu mixte, et un type d'élément nouvelle_ligne qui peut être placé à volonté dans le type d'élément lignes_texte . Pour les deux définitions de l'exemple, le Schéma de définition de types d'élément s'applique.

Le type d'élément `nouvelle_ligne` est ici un type d'élément vide. Cela est signalé par le mot clé `EMPTY` (en français: *vide*) à l'endroit où le contenu du type d'élément est défini.

Exemple d'une application valide:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE lignes_texte SYSTEM "lignes_texte.dtd">
<lignes_texte>
Ceci est le texte, mais où commence la  <nouvelle_ligne />
nouvelle ligne?
</lignes_texte>
```

Explication:

Dans l'exemple l'élément sans contenu est noté conformément aux règles pour les repères, attributs, affectations de valeur. La transcription de la signification de "nouvelle_ligne" peut naturellement n'être assurée que par un langage de style comme les CSS ou XSL.