2. XML - Eléments de syntaxe

2.1 Généralités

2.1.1 Eléments, balises et contenu

Rappel :

Un fichier **XML** contient du texte, jamais de données binaires.

Exemple :

Le code ci-dessous est un document **XML** complet comportant un seul élément *(balises + contenu)* :

```
<etudiant>
    Raymond Deubaze
</etudiant>
```

L'élément 'etudiant' comporte une **balise de début**, une **balise de fin** et un **contenu**, constitué du texte 'Raymond Deubaze' et des espaces *(blancs, tabulations, retours à la ligne)* contenus entre les balises de début et de fin.

N.B. Même si la plupart des applications choisiront d'ignorer les espaces initiaux et finaux, ceux-ci sont significatifs pour **XML**.

2.1.2 Eléments vides

En **XML** les balises de fin sont obligatoires, y compris lorsqu'un élément ne possède pas de contenu :

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

Il existe toutefois une syntaxe particulière pour refermer la balise d'un élément vide :

```
<br />
```

Les notations ci-dessus sont autorisées toutes les deux et sont strictement équivalentes.

2.1.3 Sensibilité à la casse

Contrairement à **HTML**, **XML** est sensible à la casse.

Les trois exemples ci-dessous font donc appel à trois éléments distincts :

```
<etudiant>
    Raymond Deubaze
</etudiant>
```

```
<Etudiant>
    Raymond Deubaze
</Etudiant>
```

```
<ETUDIANT>
    Raymond Deubaze
</ETUDIANT>
```

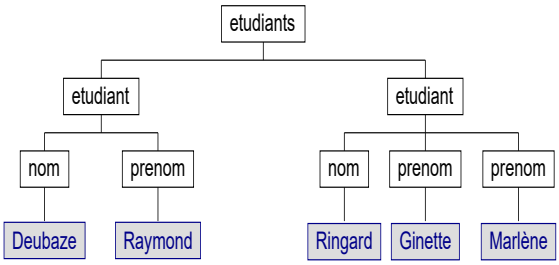
et le document ci-dessous comportera une erreur de syntaxe :

```
<etudiant>
    Raymond Deubaze
</Etudiant>
```

2.1.4 Arbre XML

Les éléments d'un document **XML** forment une structure arborescente :

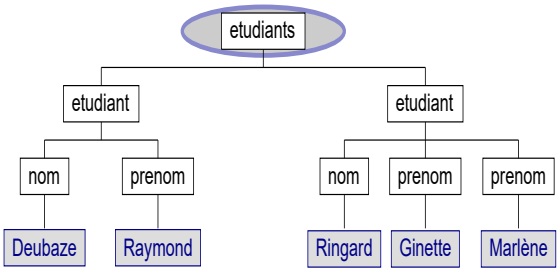
```
<etudiants>
    <etudiant>
        <nom>Deubaze</nom>
        <prenom>Raymond</prenom>
    </etudiant>
    <etudiant>
        <nom>Ringard</nom>
        <prenom>Ginette</prenom>
        <prenom>Marlène</prenom>
    </etudiant>
</etudiants>
```



La description des éléments d'un document **XML** emprunte ainsi son vocabulaire aux arbres :

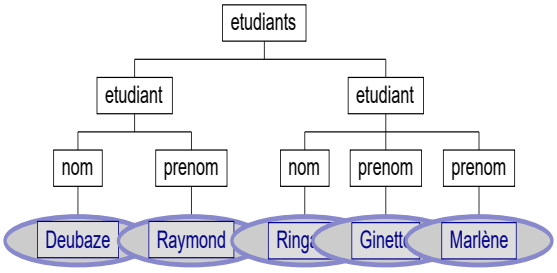
un document comporte donc un **élément racine** :

```
<etudiants>
    <etudiant>
        <nom>Deubaze</nom>
        <prenom>Raymond</prenom>
    </etudiant>
    <etudiant>
        <nom>Ringard</nom>
        <prenom>Ginette</prenom>
        <prenom>Marlène</prenom>
    </etudiant>
</etudiants>
```



et certaines feuilles de l'arbre sont constituées par le **contenu** des éléments texte :

```
<etudiants>
    <etudiant>
        <nom>Deubaze</nom>
        <prenom>Raymond</prenom>
    </etudiant>
    <etudiant>
        <nom>Ringard</nom>
        <prenom>Ginette</prenom>
        <prenom>Marlène</prenom>
    </etudiant>
</etudiants>
```



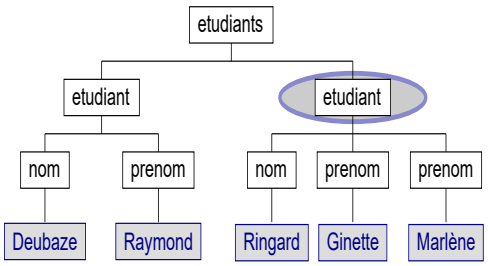
Connaissons-nous déjà d'autres éléments formant également des **feuilles de l'arbre** ?

2.1.5 La famille de l'élément courant

De la même manière qu'il est courant de se "déplacer" par l'esprit dans une arborescence de répertoires par exemple, certaines applications raisonnent en se déplaçant virtuellement dans l'arbre **XML** *(sélecteurs CSS, axes XPath, fonctions DOM, ...)*

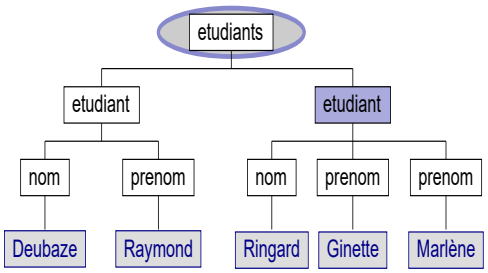
Lors d'un déplacement dans l'arbre, l'élément sur lequel on se trouve positionné à un instant donné du processus est appelé l'**élément courant** :

```
<etudiants>
    <etudiant>
        <nom>Deubaze</nom>
        <prenom>Raymond</prenom>
    </etudiant>
    <etudiant>
        <nom>Ringard</nom>
        <prenom>Ginette</prenom>
        <prenom>Marlène</prenom>
    </etudiant>
</etudiants>
```



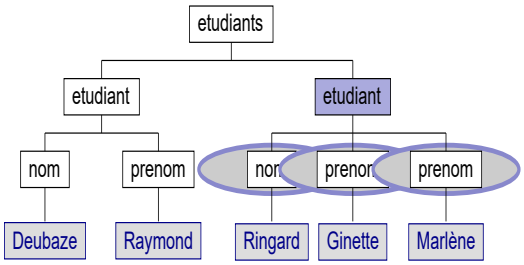
L'élément situé au-dessus de l'élément courant dans l'arbre est appelé son **élément parent** :

```
<etudiants>
    <etudiant>
        <nom>Deubaze</nom>
        <prenom>Raymond</prenom>
    </etudiant>
    <etudiant>
        <nom>Ringard</nom>
        <prenom>Ginette</prenom>
        <prenom>Marlène</prenom>
    </etudiant>
</etudiants>
```

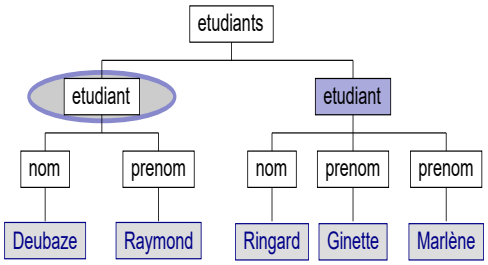


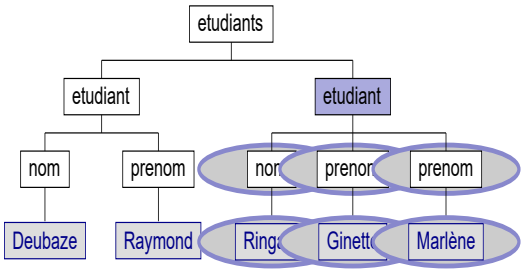
Toujours suivant la métaphore de l'arbre généalogique, les éléments situés à l'échelon juste inférieur à celui de l'élément courant sont appelés ses **enfants** :

```
<etudiants>
    <etudiant>
        <nom>Deubaze</nom>
        <prenom>Raymond</prenom>
    </etudiant>
    <etudiant>
        <nom>Ringard</nom>
        <prenom>Ginette</prenom>
        <prenom>Marlène</prenom>
    </etudiant>
</etudiants>
```



De la même manière, les éléments *(ou plus généralement les noeuds)* issus du même parent que l'élément courant seront appelés ses **frères**, l'ensemble de ceux situés plus bas dans la même branche ses **descendants** *(enfants, petits-enfants, ...)*, etc...





2.1.6 Structure d'un document XML

Les spécifications **XML** imposent qu'un document bien formé doit forcément pouvoir se mettre sous la forme d'un arbre.

Il en résulte les contraintes suivantes :

- un tel document ne peut comporter qu'**un seul élément racine**,
- les balises doivent être **correctement imbriquées**.

Voici un exemple de construction mal formée :

```
<b>ceci est <i>un exemple</b> incorrect</i> en XML
```

> En résumé :

Tout document **XML** peut se mettre sous la forme d'un arbre.

De manière duale, tout arbre peut se représenter sous la forme d'un document **XML** !

Un informaticien blasé exprimera ceci en disant :

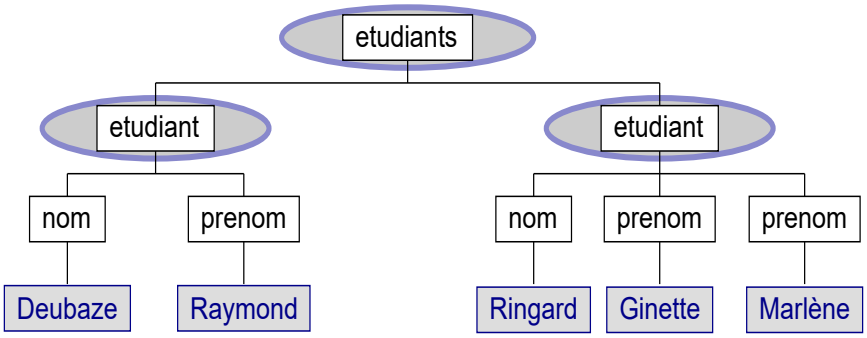
XML ? c'est juste un mécanisme pour sérialiser des arbres !

2.2 Eléments et attributs

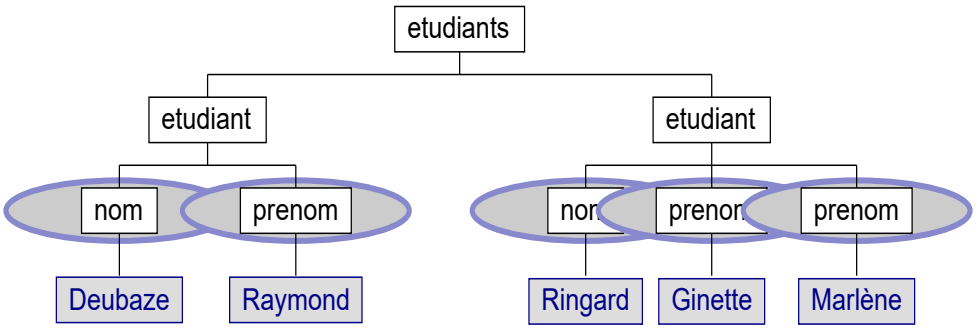
2.2.1 Contenu des éléments simples

L'arbre vu jusqu'à présent à titre d'exemple, fait apparaître deux types d'éléments simples :

- on dira que les éléments dont tous les enfants sont eux-mêmes des éléments ont un **contenu élémentaire** *(element content)* :



- tandis que les éléments qui ne contiennent que du texte seront dits à **contenu textuel** *(text content)* :

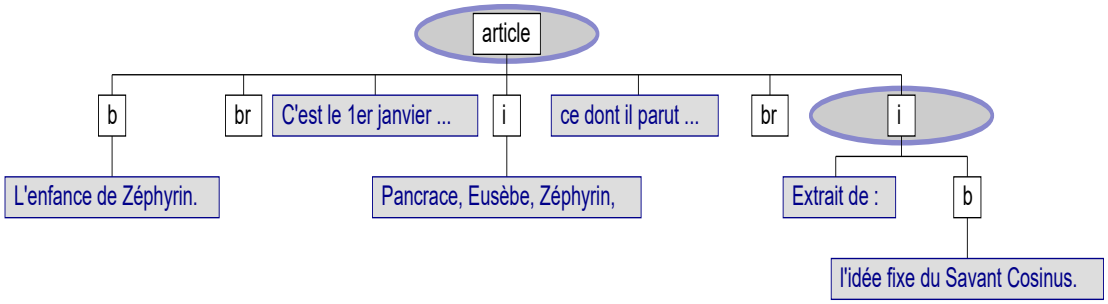


Cette structure, qui ne fait apparaître que des éléments à contenu élémentaire ou textuel est très fréquente dans le cas de **documents de données**.

2.2.2 Eléments à contenu mixte

Il existe un autre type d'éléments dits à **contenu mixte**, dont les enfants sont constitués à la fois par des éléments et du texte :

```
<article>
<b>L'enfance de Zéphyrin.</b><br/>
C'est le 1er janvier, à minuit une seconde sexagésimale de temps moyen,
que le jeune Brioché poussa ses premiers vagissements. A son baptême,
il reçut les prénoms harmonieux, poétiques et distingués de <i>Pancrace,
Eusèbe, Zéphyrin,</i> ce dont il parut se soucier comme un cloporte d'un
ophicléide.<br/>
<i>Extrait de : <b>l'idée fixe du Savant Cosinus.</b></i>
</article>
```



Les éléments à contenu mixte sont fréquents dans les **documents texte** à destination des humains *(articles, publications, pages web...)*.

2.2.3 Attributs

Outre leur contenu *(élémentaire, textuel ou mixte)* les éléments **XML** peuvent également être munis **d'attributs**.

Un **attribut** est une propriété caractérisée par un **nom**, à laquelle le document associe une **valeur**.

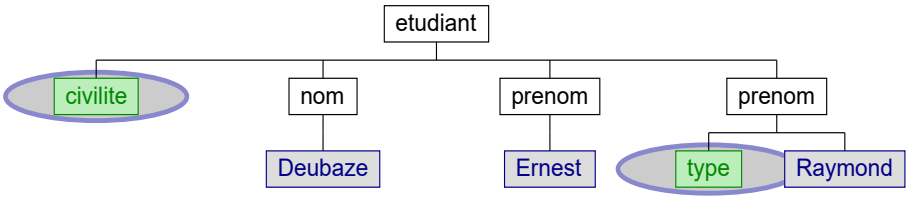
Dans l'exemple ci-dessous l'élément etudiant possède un attribut civilite dont la valeur est M, et le second élément prenom possède un attribut type dont la valeur est usuel :

```
<etudiant civilite="M">
    <nom>Deubaze</nom>
    <prenom>Ernest</prenom>
    <prenom type="usuel">Raymond</prenom>
</etudiant>
```

Ainsi qu'illustré par l'exemple ci-dessus, les règles de syntaxe pour affecter un ou plusieurs attributs à un élément au sein d'un **document XML** sont les suivantes :

- l'attribut apparait sous la forme d'une affectation nom="valeur" associée à la **balise de début** de l'élément,
- le nom de l'attribut est séparé de sa valeur par le signe **=** et d'éventuels **espaces optionnels**,
- la valeur est obligatoirement délimitée par des **apostrophes** ou des **guillemets** *(simples ou double quotes)*.

Au sein de **l'arbre XML** les attributs constituent une catégorie particulière de noeuds *(comme les noeuds texte)*, et apparaissent comme des enfants de l'élément auquel ils s'appliquent :



La **valeur** de l'attribut, comme son nom, sont des propriétés du noeud lui-même qui constitue un élément terminal de l'arbre *(feuille)*. *(N.B. contrairement à ce qu'on pourrait penser il n'y a pas de noeud texte, enfant du noeud attribut)*

2.2.4 Eléments ou attributs ?

Lors de la conception d'une application **XML** la question se pose souvent de savoir s'il faut modéliser telle ou telle information sous la forme d'un **élément** ou d'un **attribut**.

Une réponse générique n'est pas possible, et le problème est sujet à querelles de chapelles...

Ci-dessous se trouvent toutefois quelques éléments de réponse, basés sur des exemples.

> Exemple 1

Quelle solution ci-dessous faut-il par exemple privilégier lors de la conception d'une application de gestion des étudiants ?

```
<etudiant
    nom="Deubaze"
    prenom="Raymond"
/>
```

```
<etudiant>
    <nom>Deubaze</nom>
    <prenom>Raymond</prenom>
</etudiant>
```

Réponse

Un élément ne peut avoir qu'un seul attribut d'un nom donné. Il faut donc préférer les éléments aux attributs dans les cas où la propriété candidate *(ici le prénom)* serait susceptible d'avoir plusieurs valeurs.

La solution de droite est préférable, car elle laisse la porte ouverte à l'existence de plusieurs prénoms :

```
<etudiant>
    <nom>Deubaze</nom>
    <prenom>Raymond</prenom>
    <prenom>Ernest</prenom>
</etudiant>
```

> Exemple 2

Laquelle des deux solutions ci-dessous vaut-il mieux adopter ?

```
<article titre =
  "Eléments ou attributs ?"
/>
```

```
<article>
  <titre>
    Eléments ou attributs ?
  </titre>
</article>
```

Réponse

La valeur d'un attribut est forcément une chaîne de caractères, et ne pourra jamais être complétée par quoi que soit.

En s'appuyant sur un élément, celui-ci pourra posséder des attributs, et contenir lui-même d'autres éléments :

```
<article>
  <titre niveau="1"> Eléments ou attributs ?  </titre>
  <titre niveau="2"> <i>(il faut choisir)</i> </titre>
</article>
```

Complément de réponse avancé

Cet aspect est également vrai pour les **instructions de traitement** *(vues plus loin)*, couramment employées pour la génération de documents dynamiques *(cf. asp, php ou jsp)*.

A noter que la construction suivante *(courante en PHP / HTML)* est **incorrecte** du point de vue de **XML** *(mais serait-il bien nécessaire qu'elle le soit ?)* :

```
<article titre="<?php print titre_dynamique(); ?>">
```

alors que si le titre a été conçu sous forme d'un élément, la construction suivante est possible :

```
<article>
  <titre><?php print titre_dynamique(); ?></titre>
</article>
```

2.3 Entités

2.3.1 Principe des entités prédéfinies

En **XML** le caractère < *(inférieur à)* est toujours considéré comme indiquant un début de balise.

Il est donc impossible de l'utiliser tel quel dans le texte d'un document : c'est un **caractère réservé**.

La solution consiste à insérer la construction < en lieu et place du caractère désiré. En verbiage **XML** on dira qu'on fait appel à une **entité prédéfinie**.

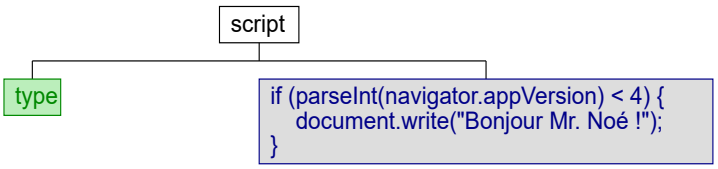
Exemple

```
<script type="text/javaScript">
    if (parseInt(navigator.appVersion) &lt; 4) {
        document.write("Bonjour Mr. Noé !");
    }
</script>
```

Techniquement les spécifications précisent que le **parseur XML** *(i.e. le module logiciel qui lit le document dans un fichier)* doit remplacer un appel d'entité par le code caractère correspondant, aussitôt que possible avant de transférer l'information en mémoire.

Il n'a pas de sens de chercher trace de l'existence d'un appel d'entité dans l'**arbre XML**.

Dans le cas ci-dessus cet arbre comprendra un élément texte dont la valeur sera une chaîne de caractères qui contiendra quelque part le caractère < :



Ceci signifie en particulier que l'application qui aura à traiter l'information ne verra pas l'appel d'entité mais bel et bien le caractère concerné.

Réciproquement, si un élément texte contient le caractère <, alors l'opération de **sérialisation** *(qui consiste à retranscrire l'arbre XML dans un document)* fera apparaître un appel d'entité.

2.3.2 Les entités prédéfinies

Il existe en **XML** deux **caractères réservés** qui donnent **obligatoirement** lieu à appel d'entité :

Caractère réservé	Appel d'entité	Description
<	<	inférieur à
&	&	esperluète

ainsi que trois **caractères spéciaux** qui peuvent être employés tels quels ou donner lieu à appel d'entité en fonction du contexte :

Caractère spécial	Appel d'entité	Description
>	>	supérieur à
'	'	guillemet simple
"	"	guillemet double

Exemples

```
<auteurs>D'Alembert &amp; al.</auteurs>
```

```
<img src='photo.png' alt='D&apos;Alembert' />
```

Les cinq entités ci-dessus sont les seules à être spécifiées comme prédéfinies. Elles sont universellement reconnues par tous les outils et applications **XML**.

N.B. Il est possible de définir ses propres appels d'entités au sein d'une **DTD** *(Définition de Type de Document)*.

2.3.3 Entités caractère

XML travaille en interne avec le codage de caractères Unicode *(ISO 10646) (i.e. les noeuds texte de l'arbre XML sont en Unicode)*.

Il est possible d'inclure n'importe quel caractère Unicode *(même non accessible au clavier)* au sein d'un **document XML**, à l'aide d'un appel d'**entité caractère**.

La syntaxe d'une **entité caractère** ressemble à celle d'une **entité prédéfinie**, à cela près que le nom de l'entité est remplacé par la caractère # *(dièse)*, suivi par le code du caractère considéré sans oublier le ; *(point-virgule)* final :

```
Insérons un espace insécable entre guillemets : ' '
```

Quelques exemples :

Entité caractère	Appel d'entité	Description
		espace insécable
©	©	copyright
®	®	marque déposée
Σ	Σ	Sigma
β	β	bêta
π	π	pi
#	∀	quel que soit
♠	♠	pique

Sachant qu'Unicode répertorie à peu près 50000 caractères différents couvrant à peu près toutes les langues du monde *(y compris le phénicien et le klingon)*, cette liste ne saurait évidemment être exhaustive...

Une variante permet d'utiliser le code caractère en hexadécimal *(au lieu de décimal)* en faisant précéder la valeur numérique par le caractère x :

```
Deux caractères α absoluments équivalents : &#945; et &#x3b1;
```

N.B. Le traitement des **entités caractère** s'effectue exactement de la même manière que celle des **entités prédéfinies**.

Il n'a toujours aucun sens de chercher trace des appels d'entité au sein de l'**arbre XML**. Le noeud texte concerné contiendra simplement le caractère visé.

2.4 Autres sections

2.4.1 Sections CDATA

Il est possible d'indiquer au sein d'un **document XML** qu'une portion de texte ne doit pas être analysée *(parsed)* à la recherche de balises ou d'appels d'entités *(parsed character data ≠ character data)*.

Cette opération s'effectue en isolant la portion de texte en question à l'aide d'une section **CDATA** :

```
<script language="JavaScript">
  <![CDATA[
    if (parseInt(navigator.appVersion) < 4) {
      document.write("Bonjour Mr. Noé !")
    }
  ]]>
</script>
```

Le texte d'une section **CDATA** se retrouvera tel quel au sein d'un noeud texte de l'**arbre XML**.

Cette construction est souvent utilisée pour intégrer des programmes *(cf. Javascript)* dans un document, ou du texte explicatif donnant des exemples de code source **XML** :

```
<p>
  Voici un exemple de code XSL :
  <exemple>
    <![CDATA[
      <xsl:template match="@email">
        <td>
          <a href="mailto:{.}">
            <xsl:apply-templates/>
          </a>
        </td>
      </xsl:template>
    ]]>
  </exemple>
  illustrant ceci.
</p>
```

Une section **CDATA** doit être correctement imbriquée avec les autres éléments.

2.4.2 Commentaires

Les **commentaires** inclus dans un document **XML** respectent la même syntaxe qu'en **HTML** :

```
<!-- ceci est un commentaire -->
```

Un commentaire ne peut pas contenir la chaîne --.

N.B. Un parseur **XML** n'est pas obligé de transmettre les commentaires à l'application. Il n'y a donc absolument aucune garantie qu'un commentaire se retrouvera au sein de l'**arbre XML**.

Il est par conséquent incorrect de baser le fonctionnement d'une application sur la présence *(ou l'absence)* d'un commentaire particulier.

Un commentaire doit être correctement imbriqué avec les autres éléments.

2.4.3 Instructions de traitement

Les **instructions de traitement** permettent de faire ce qu'il ne faut pas faire avec des commentaires : transmettre des informations à une application.

```
<?xml-stylesheet href="option.xsl" type="text/xsl" ?>
<?tic-appl-server OpticForm form init()?>
<document>
  Votre navigateur est :
  <?php return $_SERVER["HTTP_USER_AGENT"]; ?>
</document>
```

Le nom qui suit les caractères <? ouvrant la balise indique l'application à laquelle les informations sont destinées.

Du point de vue de **XML** il n'y a aucune contrainte particulière sur le texte formant le corps de la balise *(sauf caractères réservés et chaîne fermante ?>)*.

Une **instruction de traitement** doit être correctement imbriquée avec les autres éléments. La construction suivante *(courante en PHP / HTML)* est donc **incorrecte** :

```
<article titre="<?php print titre_dynamique(); ?>">
```

2.4.4 Déclaration XML

Un document **XML** peut commencer par une **déclaration XML** *(présence non obligatoire)*.

```
<?xml version="1.0" ?>
```

La **déclaration XML** peut mentionner le jeu de caractères du document :

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

D'après les spcécifications, les seuls jeux de caractères obligatoirement reconnus par un parseur **XML** sont **UTF-8** et **UTF-16**.

Toutefois, les parseurs reconnaissent en général d'autres jeux de caractères *(point fort des parseurs commerciaux)*, et en particulier **ISO-8859-1** qui nous a longtemps concernés en premier chef *(cf. caractères accentués français)*, mais est actuellement remplacé par **UTF-8** par les systèmes récents.

2.5 Vers des documents bien formés

2.5.1 Noms autorisés

Les noms d'éléments et d'attributs peuvent contenir n'importe quel caractère alphanumérique *(A-Z a-z 0-9)* ainsi que quelques caractères de ponctuation : _ *(souligné)*, - *(tiret)*, . *(point)* et : *(double-point)* qui sera réservé pour les espaces de noms *(namespaces)*.

Un **nom XML** ne peut pas commencer par un chiffre, un tiret ou un point.

Tous les autres caractères *(espaces, apostrophes, $, virgule, point-virgule...)* sont interdits.

Les caractères locaux *(cf. caractères accentués en français)* sont acceptés dans la mesure où la **déclaration XML** indique le type de codage utilisé pour le document et que celui-ci est reconnu par le parseur.

2.5.2 Noms autorisés ?

Saurez-vous localiser les erreurs ?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<exemples>
    <Numéro-SS>30363112089</Numéro-SS>
    <adresse_postale>Av. Guy de Collongue</adresse_postale>
    <nom_d'emprunt>Johnny</nom_d'emprunt>
    <mois-année>12/99</mois-année>
    <jours/mois>01/02</jours/mois>
    <nb:>ne pas oublier</nb:>
    <2-vous-à-moi>petit à parté</2-vous-à-moi>
</exemples>
```

2.5.3 Documents bien formés

Un document qui respecte les règles de la syntaxe **XML** est appelé un **document XML bien formé**.

Voici le récapitulatif de quelques règles simples participant à l'élaboration de documents **bien formés** :

- a toute balise de début doit correspondre une balise de fin,
- les balises doivent être correctement imbriquées,
- il ne peut y avoir qu'un seul élément racine,
- la valeur des attributs doit se trouver entre guillemets,
- un élément ne peut avoir deux attributs du même nom,
- les commentaires et instructions de traitement doivent être correctement imbriqués avec les balises,
- les caractères < et & ne doivent pas apparaître tels quels.

Un **parseur XML** conforme aux spécifications **ne doit pas** transmettre un document mal formé à l'application.

> Comment vérifier si un document est bien formé ?

Pour cela il suffit de l'ouvrir avec un navigateur de dernière génération *(Firefox, IE 6.0, Mozilla 1.5, NS 7.0 ...)* :

Exemple de document bien formé :
[http://dmolinarius.github.io/demofiles/mod-84/xml/xml/exemple_1.xml]
[http://dmolinarius.github.io/demofiles/mod-84/xml/xml/exemple_erreur.xml]