

## 4. Définition de Type de Document

### 4.1 Déclaration de Type de Document

#### 4.1.1 Déclaration de Type de Document

Un document **XML** fait référence à une **DTD** grâce à une **déclaration de type de document** placée dans le prologue du document (*i.e. après la déclaration XML, mais avant l'élément racine*).

La syntaxe générale d'une déclaration de type de document est :

```
<!DOCTYPE element_racine SYSTEM "URL_dtd">
```

##### > Exemple

La déclaration ci-dessous indique que le nom de l'élément racine est "*classe*" et que la **DTD** peut être trouvée à l'URL "*classe-primaire.dtd*".

```
<?xml version="1.0" ?>
<!DOCTYPE classe SYSTEM "classe-primaire.dtd">
<classe>
.
.
.
</classe>
```

#### 4.1.2 DTD publique

Lorsqu'un document est muni d'une **DTD**, les applications qui accèdent au document sont susceptibles de devoir accéder également à la **DTD**, aux fins de validation, pour récupérer la valeur par défaut de certains attributs, ou obtenir la valeur de certaines entités.

Dans le cas d'une application standard (*i.e. très répandue*), on peut déclarer la **DTD** grâce à un **identifiant public** :

```
<!DOCTYPE element_racine PUBLIC "URI_public" "URL_dtd">
```

L'identifiant public est un **URI**, il s'agit donc d'une chaîne globalement unique qui identifie l'application.

Ceci permet éventuellement à un processeur qui reconnaît l'identifiant public de ne pas ouvrir de connexion réseau pour accéder à la **DTD**, car il la connaît par construction (*cf. navigateurs dans le cas de XHTML*).

##### > Exemple

La déclaration ci-dessous indique qu'il s'agit d'un document **XHTML** :

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

#### 4.1.3 DTD interne

Une déclaration de type de document ne fait pas forcément référence à un document externe.

La **DTD** peut être développée directement au sein de sa déclaration :

```
<!DOCTYPE classe [
  <!ELEMENT classe (promo,etudiant+)>
  <!ELEMENT etudiant (nom, prenom)>
  <!ELEMENT nom (#PCDATA)>
  <!ELEMENT prenom (#PCDATA)>
  <!ELEMENT promo (#PCDATA)>
]>
```

Cette technique est particulièrement utile en phase de développement, lorsqu'on met au point à la fois le premier document d'une série et la **DTD** qui les décrit, de manière à n'avoir qu'un seul fichier source à maintenir.

Les deux formats (*DTD interne et externe*) peuvent être associés dans la déclaration :

```
<!DOCTYPE classe SYSTEM "promo-centrale.dtd" [
  <!ELEMENT classe (promo,etudiant+)>
]>
```

Dans cet exemple, la plupart des éléments sont déclarés au sein de la **DTD externe** "*promo-centrale.dtd*", alors que l'élément racine "*classe*" est déclaré localement.

**N.B.** Les sous-ensembles **externe** et **interne** de la **DTD** doivent être compatibles. On ne peut pas, de cette manière, surcharger des déclarations d'éléments ou d'attributs. Seules les déclarations d'entités peuvent être surchargées.

#### 4.1.4 Déclaration standalone

Lorsqu'une déclaration de type de document fait référence à une **DTD** (ou un sous-ensemble) externe, il est conseillé de l'indiquer au sein de la **déclaration XML** à l'aide du pseudo-attribut "*standalone*" :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE classe SYSTEM "exemple_2.dtd">
```

Cette mention sert à indiquer à un **parseur non validant** qu'il devrait tout de même ouvrir et analyser la **DTD** qui est susceptible de contenir par exemple la valeur par défaut de certains attributs.

**N.B.** De nombreux parseurs n'analysent pas les sous-ensembles de **DTD** externes, et il est toujours possible de configurer les parseurs (*même validants*) pour ne pas le faire. Dans ce cas, le résultat obtenu peut dépendre du parseur ou de sa configuration.

## 4.2 Principe d'une DTD

Une **DTD** permet de définir le vocabulaire (*noms des éléments, attributs et entités*) et la grammaire (*contexte dans lequel peuvent ou doivent apparaître ces noms*) d'un **document XML**.

Une **DTD** est essentiellement composée de **déclarations de balisage** qui se subdivisent en :

- déclarations de type d'éléments,
- déclarations de liste d'attributs,
- déclarations d'entités.

L'extrait ci-dessous donne un aperçu de déclaration de type d'élément, de liste d'attributs, et d'entité :

```
<!ELEMENT classe (promo,etudiant+)>
. . .
<!ATTLIST etudiant civilité ( M | F ) #REQUIRED>
. . .
<!ENTITY contact "Daniel.Muller@ec-lyon.fr">
```

## 4.3 Déclaration de type d'élément

### 4.3.1 Principe

La **déclaration de type** d'un élément précise le **modèle de contenu** de cet élément. Aucun type d'élément ne peut être déclaré plus d'une fois.

La syntaxe générale d'une déclaration de type d'élément est :

```
<!ELEMENT nom_element modele_de_contenu >
```

Le modèle de contenu d'un élément simple peut prendre l'une des formes suivantes :

**EMPTY** - l'élément est vide (*comme par exemple les balises HR, BR, IMG ... en langage XHTML*),

```
<!ELEMENT img EMPTY>
```

**ANY** - l'élément peut contenir n'importe quel autre élément déclaré (*très rarement employé*),

```
<!ELEMENT test ANY>
```

**(#PCDATA)** - l'élément est un élément texte (*Parsed Character Data*), il ne peut contenir de sous-élément,

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

**(liste de sous-éléments)** - la liste donne les noms (*séparés par des virgules et des blancs facultatifs*) et l'ordre des sous-éléments autorisés :

```
<!ELEMENT étudiant (nom,prénom)>
```

Dans le cas de l'exemple ci-dessus, un élément "*étudiant*" devra obligatoirement comporter un (*et un seul*) sous-élément "*nom*", suivi par un (*et un seul*) sous-élément "*prénom*", dans cet ordre.

### 4.3.2 Modèles de contenu élémentaire

Dans le cas des éléments simples à contenu élémentaire (*i.e. que des sous éléments, pas de texte*) on peut spécifier le type des sous-éléments, leur ordre et dans une certaine mesure le nombre d'occurrences autorisées.

Pour cela, le **modèle de contenu** peut comporter :

- **des listes** - avec l'opérateur ";",
- **des alternatives** - avec l'opérateur "|",
- **des groupes** - entre parenthèses "()".

Les éléments ou groupes d'éléments peuvent être post-fixés par un opérateur indiquant le nombre d'occurrences autorisées :

- **+** - une ou plusieurs occurrences,
- **\*** - zéro ou plusieurs occurrences,
- **?** - zéro ou une occurrence.

#### > Exemples

La déclaration ci-dessous indique qu'une classe est composée de un ou plusieurs étudiants :

```
<!ELEMENT classe (étudiant+)>
```

Un étudiant doit avoir un nom, suivi par un ou plusieurs prénoms :

```
<!ELEMENT étudiant (nom,prénom+) >
```

Il est possible de mentionner son numéro de sécurité sociale :

```
<!ELEMENT étudiant (nom,prénom+,insee?) >
```

A défaut, on peut donner son identifiant interne ECL :

```
<!ELEMENT étudiant (nom,prénom+, (insee|eclid)?) >
```

Il est éventuellement inscrit à des cours :

```
<!ELEMENT étudiant (nom,prénom+, (insee|eclid)?,cours*) >
```

**N.B.** L'ordre des listes est obligatoire. Pour autoriser des ordres différents, la seule solution consiste à lister toutes les alternatives autorisées :

```
<!ELEMENT étudiant ((nom,prénom+) | (prénom+,nom)) >
```

Comment pourrait-on autoriser également le nom à apparaître entre deux prénoms ?

Exemple :

<http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/elt-elt-content-exemple.html>

### 4.3.3 Modèles de contenu mixte

Pour les éléments à contenu mixte (*texte mêlé à des sous-éléments*), il est possible de préciser le type des sous-éléments autorisés, mais pas leur ordre ni le nombre de leurs occurrences.

La syntaxe du modèle de contenu est la même que dans le cas des contenus élémentaires purs, sauf qu'il faut obligatoirement mentionner le texte en première position à l'aide du mot-clé **"#PCDATA"** (*Parsed Character Data*).

De plus, le modèle de contenu comporte obligatoirement l'opérateur **"\*"** :

```
<!ELEMENT paragraphe (#PCDATA | gras | italiques)* >
<!ELEMENT gras (#PCDATA | italiques)* >
<!ELEMENT italiques (#PCDATA | gras)* >
```

## 4.4 Déclaration de liste d'attributs

### 4.4.1 Principe

La **déclaration de liste d'attributs** d'un élément sert à :

- définir un **jeu d'attributs** pour les éléments de ce type,
- établir un **modèle de contenu** de ces attributs,
- fournir une **valeur par défaut** pour certains attributs.

S'il existe plus d'une définition pour un même attribut d'un type d'élément donné, seule est prise en compte la première, les déclarations subséquentes sont ignorées.

La syntaxe générale d'une déclaration de liste d'attribut est :

```
<!ATTLIST nom_element (nom_attribut modele_de_contenu valeur_par_defaut)+ >
```

La déclaration ci-dessous indique par exemple que les éléments du type "*étudiant*" possèdent un attribut obligatoire nommé "*nom*", du type "*CDATA*" (*Character Data*) c'est à dire simplement composé de texte (*non analysé*) :

```
<!ATTLIST étudiant nom CDATA #REQUIRED >
```

#### 4.4.2 Liste d'attributs

S'il existe plus d'une déclaration de liste d'attributs pour un type d'élément donné, le contenu de toutes les déclarations fournies est fusionné.

Lorsqu'un même élément possède plusieurs attributs, ceux-ci peuvent donc faire l'objet d'une seule ou de plusieurs déclarations :

```
<!ATTLIST étudiant
  nom CDATA #REQUIRED
  prénom CDATA #REQUIRED
  civilité ( M | F ) #REQUIRED
  statut ( célibataire | marié | mariée ) "célibataire" >
```

est équivalent à :

```
<!ATTLIST étudiant nom CDATA #REQUIRED >
<!ATTLIST étudiant prénom CDATA #REQUIRED >
<!ATTLIST étudiant civilité ( M | F ) #REQUIRED >
<!ATTLIST étudiant statut ( célibataire | marié | mariée ) "célibataire" >
```

**N.B.** Lorsqu'un même attribut est autorisé avec les mêmes caractéristiques pour plusieurs éléments (*cf. par exemple en XHTML, les attributs communs à TD et TH*), il doit être **répété** pour chacun des éléments.

```
<!ATTLIST étudiant nom CDATA #REQUIRED >
<!ATTLIST étudiant prénom CDATA #REQUIRED >
<!ATTLIST professeur nom CDATA #REQUIRED >
<!ATTLIST professeur prénom CDATA #REQUIRED >
```

Cette lourdeur peut être contournée par l'utilisation d'**entités paramètres** (*vues plus loin*).

#### 4.4.3 Types d'attributs

Contrairement aux éléments, dont le contenu textuel ne peut être contraint à l'aide d'une **DTD**, il est possible dans une certaine mesure de restreindre le domaine des valeurs possibles pour un attribut.

Ci-dessous les modèles de contenu les plus courants :

**CDATA** - chaîne de caractères (*cf. l'attribut src de l'élément img en XHTML*),

```
<!ATTLIST img src CDATA #REQUIRED>
```

```

```

**NMTOKEN** (*Name Token*) - chaîne de caractères sans espaces ressemblant à un nom **XML**, à ceci près que tous les caractères autorisés sont permis pour l'initiale (*cf. par exemple l'attribut name des hyperliens en XHTML*),

```
<!ATTLIST a name NMTOKEN #IMPLIED>
```

```
<a name="2B3"> . . . </a>
```

**NMTOKENS** - liste de **NMTOKEN** séparés par des espaces. Soit par exemple un élément *"Allow"* qui admet un attribut *"host"* composé d'une liste de numéros IP séparés par des espaces :

```
<!ATTLIST Allow host NMTOKENS #REQUIRED>
```

```
<Allow host="156.18.10.1 156.18.10.2"/>
```

**énumération** - la valeur est à prendre parmi une liste de **NMTOKEN** fournie au sein de la déclaration (cf. l'attribut *align* de l'élément *img* en *XHTML*),

```
<!ATTLIST img align (top|middle|bottom) #IMPLIED>
```

```
<img align="middle" ... >
```

#### 4.4.4 Identifiants uniques

Il existe une catégorie particulière d'attributs, dont la valeur doit correspondre à un identifiant unique au sein d'un même document. Ces attributs sont déclarés avec un modèle de contenu particulier :

**ID - nom XML**, identifiant unique (cf. les attributs *id* en *XHTML*) :

```
<!ATTLIST span id ID #IMPLIED>
```

```
<span id="drop_zone"> . . . </span>
```

**N.B.** Si, au sein d'un document, deux éléments ont la même valeur pour deux attributs (*même de nom différent*) déclarés du type **ID**, alors le document est non valide.

Il est également possible de contraindre certains attributs à correspondre à un identifiant, ou à une liste d'identifiants présents dans le document :

**IDREF - nom XML** donnant la valeur d'un attribut du type **ID** d'un élément du même document.

```
<!ATTLIST toc-entry link-to IDREF #IMPLIED>
```

```
<toc-entry link-to="chap1">Chapitre 1</toc-entry>
. . .
<chapter id="chap1">
. . .
</chapter>
```

**IDREFS** - une liste d'**IDREFs** séparés par des espaces :

```
<!ATTLIST login accesslist IDREFS #REQUIRED>
```

```
<role id="admin"> . . . </role>
<role id="webmaster"> . . . </role>
<role id="baseroor"> . . . </role>
. . .
<login accesslist="admin webmaster baseroor">administrateur</login>
```

Exemple :

<http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/atr-id-exemple.html>

**N.B.** Il faut remarquer que ce mécanisme reste très limité, puisqu'il ne permet pas d'indiquer à quel type d'élément on veut faire référence ni si la référence doit être unique.

#### 4.4.5 Valeur par défaut

Une déclaration de liste d'attributs, outre les noms et les modèles de contenu, indique également si l'attribut est obligatoire ou non, et quel comportement adopter lorsqu'il est absent.

Ci-dessous les quatre cas possibles :

**#IMPLIED** - l'attribut est optionnel, il n'y a pas de valeur par défaut,

```
<!ATTLIST img width CDATA #IMPLIED
             height CDATA #IMPLIED>
```

```
<img width="100%" ... >
```

**#REQUIRED** - la présence de l'attribut est obligatoire, il n'y a pas de valeur par défaut,

```
<!ATTLIST img src CDATA #REQUIRED>
```

```

```

**"une valeur par défaut"** - l'attribut est optionnel, mais lorsqu'il est absent le parseur doit se comporter comme s'il était présent avec la valeur par défaut fournie,

```
<!ATTLIST input type (text|password|checkbox|radio|hidden) "text">
```

```
<input ... />
<input type="text" ... />
<input type="radio" ... />
```

**#FIXED "valeur"** - l'attribut est optionnel, s'il est absent le parseur doit se comporter comme s'il était présent avec la valeur par défaut fournie, s'il est présent il ne peut prendre que la valeur fournie.

```
<!ATTLIST html xmlns CDATA #FIXED "http://www.w3.org/1999/xhtml">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/atr-value-exemple.html>]

#### 4.4.6 Quelques exemples

```
<!ATTLIST li type (disc|square|circle) #IMPLIED >
```

L'attribut **"type"** de l'élément **"li"** est optionnel. S'il est présent, il peut prendre une des valeurs **"disc"**, **"square"** ou **"circle"**. S'il est absent il n'y a pas de valeur par défaut (*i.e. il n'y aura pas d'attribut type dans l'arbre XML en mémoire*).

```
<!ATTLIST li type (disc|square|circle) "disc" >
```

L'attribut **"type"** de l'élément **"li"** est optionnel. S'il est présent, il peut prendre une des valeurs **"disc"**, **"square"** ou **"circle"**. S'il est absent **le parseur** doit se comporter **comme s'il était présent** et le transmettre à l'application avec la valeur **"disc"**.

```
<!ATTLIST html xmlns CDATA #FIXED 'http://www.w3.org/1999/xhtml'>
```

L'attribut *"xmlns"* de l'élément *"html"* est optionnel. S'il est présent, il ne peut prendre que la valeur précisée. Présent ou absent, **le parseur** doit se comporter strictement comme s'il était présent avec la valeur ci-dessus et le transmettre à l'application.

```
<!ATTLIST meta content CDATA #REQUIRED>
```

L'attribut *"content"* de l'élément *"meta"* est obligatoire. Il n'y a pas de valeur par défaut.

## 4.5 Déclarations d'entités

### 4.5.1 Entité générale

Une **entité** est une sorte de macro, pour représenter un caractère ou une chaîne de caractères par un nom.

Une **entité générale** est une macro qui s'emploie au sein du **document XML**.

La syntaxe d'une déclaration d'entité générale est :

```
<!ENTITY nom_entite "valeur_entite" >
```

Au sein du document l'utilisation se fait comme pour les **entités prédéfinies**, en faisant précéder le nom de l'entité par une esperluète *"&"* et suivre par un point-virgule *","* :

```
&nom_entite;
```

#### > Exemple 1

Ci-dessous un exemple de **déclaration d'entité** pour les caractères *"blanc insécable"* et *"e accent aigu"*.

```
<!ENTITY nbsp "&#160;"> <!-- no-break space -->
<!ENTITY eacute "&#233;"> <!-- small e with acute -->
```

```
<p>un&nbspbsp;&lt;&eacute;l&eacute;ment&gt;</p>
```

Tester un exemple :

[\[http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-general-exemple1.html\]](http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-general-exemple1.html)

#### > Exemple 2

```
<!DOCTYPE page SYSTEM "doctype.dtd" [
  <!ENTITY contact "Daniel.Muller@ec-lyon.fr">
]>
```

```
<a href="mailto:&contact;">votre contact</a>
```

Remarquer ci-dessus comment l'entité est déclarée dans la partie interne de la déclaration de type de document, et appelée comme valeur d'un attribut.

Tester un exemple :

[\[http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-general-exemple2.html\]](http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-general-exemple2.html)



### > Exemple 3

Dans la mesure où le résultat n'est pas récursif, la déclaration d'une entité générale peut invoquer la valeur d'une autre entité générale :

```
<!ENTITY contact "Daniel.Muller">
<!ENTITY address ' <a href="mailto:&contact;">&contact;</a>' >
```

Tester un exemple :

[\[http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-general-exemple3.html\]](http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-general-exemple3.html)

### 4.5.2 Entité générale externe analysée

La valeur d'une entité générale peut être stockée dans un document à part. On parlera alors d'**entité générale externe analysée** (*parsed external general entity*).

La déclaration se fait sur l'un des modèles suivants (*entité système ou publique*) :

```
<!ENTITY nom_entite SYSTEM "URL_entite">
<!ENTITY nom_entite PUBLIC "URI_public" "URL_entite">
```

L'insertion de l'entité au sein du document se fait de manière classique :

```
&nom_entite;
```

**N.B.** Le contenu d'une entité externe analysée est très similaire à celui d'un document XML. Un tel document peut comporter (*ou non*) une **déclaration XML**.

La seule différence est qu'une entité externe analysée peut comporter **plusieurs éléments racine**.

### > Exemple

Lorsque l'information contenue dans un même document **XML** devient très volumineuse, il est beaucoup plus pratique de la manipuler sous forme de fichiers multiples, correspondant par exemple aux divers chapitres (*voire aux sections*) d'un livre.

Les divers chapitres seront alors déclarés au sein du document principal en tant qu'entités externes :

```
<!DOCTYPE bouquin SYSTEM "livre.dtd" [
  <!ENTITY intro SYSTEM "introduction.ent">
  <!ENTITY chap1 SYSTEM "chapitre_1.ent">
  <!ENTITY chap2 SYSTEM "chapitre_2.ent">
  <!ENTITY concl SYSTEM "conclusion.ent">
]>
```

Et le contenu du document sera composé à l'aide d'appels d'entités :

```
<bouquin>
  &intro;
  &chap1;
  &chap2;
  &concl;
</bouquin>
```

Tester un exemple :

[\[http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-parsed-ext-exemple1.html\]](http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-parsed-ext-exemple1.html)

### 4.5.3 Entité paramètre

Contrairement à une entité générale qui s'emploie au sein du contenu XML lui-même, une **entité paramètre** s'emploie au sein de la **DTD**.

La définition d'une entité paramètre se fait conformément au modèle :

```
<!ENTITY % nom_entite "valeur_entite" >
```

La syntaxe permettant d'invoquer une entité paramètre (*au sein de la DTD*) est similaire à celle permettant d'appeler une entité générale (*au sein du code XML*). Il suffit de remplacer l'esperluète par un caractère "%":

```
%nom_entite;
```

#### > Exemple - Liste d'attributs

Les entités paramètre sont très souvent utilisées au sein des **DTD** pour définir des attributs ou des listes d'attributs communes à plusieurs éléments :

```
<!ENTITY % usemap "usemap CDATA #IMPLIED">
<!ATTLIST img %usemap; >
<!ATTLIST input %usemap; >
<!ATTLIST object %usemap; >
```

Elles servent également à des fins mnémotechniques, afin de rendre les **DTD** complexes plus lisibles (*noter comment la définition d'une entité peut faire elle-même appel à une entité*).

```
<!ENTITY % StyleSheet "CDATA">
  <!-- style sheet data -->
<!ENTITY % Text "CDATA">
  <!-- used for titles etc. -->
<!ENTITY % coreattrs
  "id          ID          #IMPLIED
   class       CDATA      #IMPLIED
   style       %StyleSheet; #IMPLIED
   title       %Text;      #IMPLIED"
  >
```

```
<!ATTLIST b %coreattrs >
<!ATTLIST i %coreattrs >
<!ATTLIST em %coreattrs >
. . .
```

#### > Exemple - Déclaration d'élément

```
<!ENTITY % head.misc
  "(script|style|meta|link|object|isindex) *">
<!ELEMENT head (%head.misc;,
  ((title, %head.misc;, (base, %head.misc;)? ) |
   (base, %head.misc;, (title, %head.misc;))))>
```

Le déclaration ci-dessus signifie que l'élément **"head"** contient un élément **"titre"** unique et un élément **"base"** optionnel dans un ordre quelconque avec une combinaison éventuellement vide des éléments de **"head.misc"**.

#### > Exemple - DTD complexe

Les **DTD** d'applications **"réelles"** qui atteignent une certaine complexité font un usage intensif des entités paramètres.

 Consulter la DTD de XHTML Strict

[\[https://www.w3.org/TR/xhtml1/DTDs.html#a\\_dtd\\_XHTML-1.0-Strict\]](https://www.w3.org/TR/xhtml1/DTDs.html#a_dtd_XHTML-1.0-Strict)

#### 4.5.4 Entité paramètre externe

Les entités paramètres, comme les entités générales, sont susceptibles de faire l'objet d'un document séparé, public ou privé, connu par un **URI** et/ou accessible par une **URL** :

```
<!ENTITY % nom_entite SYSTEM "URL_entite">
<!ENTITY % nom_entite PUBLIC "URI_public" "URL_entite">
```

L'insertion de l'entité au sein de la **DTD** se fait toujours de la même manière :

```
%nom_entite;
```

Le code ci-dessous indique comment réutiliser les entités générales correspondant aux caractères **Latin1** naturellement accessibles en **XHTML** depuis un document **XML** de notre composition :

```
<!ENTITY % HTMLlat1 PUBLIC
    "-//W3C//ENTITIES Latin 1 for XHTML//EN"
    "xhtml-lat1.ent">
%HTMLlat1;
```

🔗 Voir l'entité externe xhtml-lat1.ent

[[https://www.w3.org/TR/xhtml1/dtds.html#a\\_dtd\\_Latin-1\\_characters](https://www.w3.org/TR/xhtml1/dtds.html#a_dtd_Latin-1_characters)]

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/xml/dtd/ent-param-ext-exemple.html>]

## 4.6 Espaces de noms

La **DTD** d'un document utilisant des espaces de noms doit définir les éléments à l'aide de leur nom qualifié (*Qname*) comportant le préfixe et la partie locale.

```
<!ATTLIST html xml:lang NMTOKEN #IMPLIED>
```

## 4.7 Validation

### > Validateurs en ligne

Comme le montrent les exemples de ce cours, les navigateurs se contentent en général de documents bien formés, et n'effectuent pas l'étape de validation.

Pour un usage occasionnel il existe des services de validation en ligne :

🔗 <http://validator.w3.org/>

[<http://validator.w3.org/>]

🔗 <http://www.cogsci.ed.ac.uk/~richard/xml-check.html>

[<http://www.cogsci.ed.ac.uk/~richard/xml-check.html>]

### > IDEs

Pour un usage plus intensif il sera plus utile d'utiliser un environnement de développement intégré, dédié à XML (*liste subjective et non exhaustive*) :

🔗 XML Copy Editor

[<http://xml-copy-editor.sourceforge.net/>]

🔗 XML Notepad

[<https://xmlnotepad.codeplex.com/>]

🔗 XML Spy


[<http://www.altova.com/simpliedownload2c.html>]

### > APIs

Enfin, pour les plus geeks, il existe des APIs pour valider des documents XML depuis à peu près n'importe quel langage. Par exemple :

 java

[<http://docs.oracle.com/javase/7/docs/api/javax/xml/validation/package-summary.html>]

 Perl

[<http://search.cpan.org/~bbc/XML-Validate-1.025/lib/XML/Validate.pm>]

 Python

[<http://lxml.de/validation.html>]