---

3. Les DTD

3.1 Introduction aux DTD de XML

- DTD = Document Type Definition
- Note: Il existe d'autres formalismes pour définir une grammaire, par ex.:
  - XSD (XML Schema Definition) ou RELAX NG

Une DTD est une grammaire qui definit:

1. 1. les balises (tags) possibles et leurs attributs
2. 2. L'imbrication des balises à l'intérieur d'autres balises
3. 3. Quels balises et attributs sont à option et lesquels sont obligatoires

Chaque balise XML est défini une seule fois comme un élément dans la DTD

- Exemple illustratif:

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

3.2 Association d'une DTD avec un fichier XML

A. Déclaration d'une DTD dans un fichier XML

Exemple:

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

<!DOCTYPE hello SYSTEM "hello.dtd">
```

Il existe 4 façons d'utiliser une DTD

1. On ne déclare pas de DTD (dans ce cas le fichier est juste "bien formé")
2. On déclare la DTD et on y ajoute les définitions dans le même fichier (DTD interne)
   - On parle dans ce cas d'un XML "standalone" (le fichier XML se suffit à lui-même)
3. On déclare la DTD en tant que DTD "privée", la DTD se trouve quelque part dans votre système ou sur Internet
   - répandu pour les DTDs "faites maison"
4. On déclare une DTD "public", c.a.d. on utilise un nom officiel pour la DTD.
   - cela présuppose que votre éditeur et votre client connaissent cette DTD
   - répandu pour les DTDs connues comme XHTML, SVG, MathML, etc.

Lieu de la déclaration

- La DTD est déclarée entre la déclaration de XML et le document lui-même.
- La déclaration de XML et celle de la DTD font parti du prologue
  - (qui peut contenir d'autres éléments comme les processing instructions)
- Attention: l'encodage de la DTD doit correspondre à celui des fichiers XML !

B. Syntaxe de la déclaration

- Chaque déclaration de la DTD commence par:

```
  <!DOCTYPE
```

- ... et fini par:

```
  >
```

- La racine de l'arbre XML (ici: <*hello*>) doit être indiquée après <!DOCTYPE
- Syntaxe pour définir une DTD interne (seulement !)
  - La DTD sera insérée entre [...]

```
   <!DOCTYPE hello [

       <!ELEMENT hello (#PCDATA)>

                   ]
>
```

- Syntaxe pour définir une DTD privée externe:
  - La DTD est dans l'URL indiqué après le mot clef "SYSTEM".

<!DOCTYPE hello SYSTEM "hello.dtd">

  - Important: Ne pas répéter la déclaration de la DTD dans le fichier *.dtd !!
  - Dit autrement: La déclaration de la DTD se fait dans le fichier XML et PAS dans le fichier *.dtd. Ce fichier ne définit que les règles ...

C. Définition de la racine de l'arbre

- Le mot *"hello"* après le mot clef *DOCTYPE* indique que "hello" est l'élément racine de l'arbre XML.

<!DOCTYPE *hello* SYSTEM "hello.dtd">

- Important: la racine de l'arbre XML doit être définie comme ELEMENT dans la DTD
  - normalement on la met au début, mais ce n'est pas une obligation

Quelques exemples:

Exemple 3-1: Hello XML sans DTD

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

<hello> Hello XML et hello cher lecteur ! </hello>
```

Exemple 3-2: Hello XML avec DTD interne

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

<!DOCTYPE hello [

   <!ELEMENT hello (#PCDATA)>

   ]>

<hello> Hello XML et hello chère lectrice ! </hello>
```

Exemple 3-3: Hello XML avec DTD externe

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

<!DOCTYPE hello SYSTEM "hello.dtd">

<hello> Hello XèMèLè et hello cher lectrice ! </hello>
```

Exemple 3-4: Un fichier RSS (DTD externe public)

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

<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"

 "http://my.netscape.com/publish/formats/rss-0.91.dtd">

<rss version="0.91">

<channel> ...... </channel>

</rss>
```

3.3 Déclaration d'éléments (tags)

Rappel du principe:

- On doit définir *chaque élément* que l'on pense utiliser dans les documents
- On doit indiquer comment les éléments s'imbriquent
- Chaque élément ne se définit qu'une seule fois !!

Exemple de sensibilisation:

```
<!ELEMENT name (family,given)>

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

veut dire que:

  - l'élément "name" a deux éléments enfants: family et given
  - l'élément "family" ne contient que du texte (pas d'autres éléments)

En utilisant un "schéma":

```
name            ==>     family + given

family          ==> "texte"
```

Syntaxe de la définition d'un élément:

<!ELEMENT nom_balise spécification_contenu>

La spécification du contenu d'un élément contient:

  - soit une combinaison d'autres éléments,
  - soit une combainaison d'autres élément plus des éléments spéciaux #PCDATA, ANY (contenu mixte)
  - soit l'élément #PCDATA, ou ANY
  - soit EMPTY

Règles de combinaison: On peut combiner selon les règles ci-dessous:

A et B = tags	Explication specification_contenu	Exemples
A?	A (un seul) est une option, (donc: A ou rien)	`<!ELEMENT person (name,email? )`
A+	Il faut un ou plusieurs A	`<!ELEMENT person (name,email+ )`
A*	A est une option, il faut zèro, un ou plusieurs A	`<!ELEMENT person (name,email* )`
A | B	Il faut A ou B, mais pas les deux	`<!ELEMENT person (email | fax )`
A , B	Il faut A, suivi de B (dans l'ordre)	`<!ELEMENT person (name ,email? )`
(A, B) +	Les parenthèses regroupent. Ici: un ou plusieurs (A suivi de B)	`<!ELEMENT liste ( name,email)+`

Eléments spéciaux

Elément spéciaux	Explication specification_contenu	Exemples
#PCDATA	"Parsed Character Data" Données (non-interprétées par XML) dans le langage d'encodage courant.	`<!ELEMENT email (#PCDATA)>`
ANY	Mot clé qui indique que tous les éléments sont autorisés (déconseillé)	`<!ELEMENT person ANY>`
EMPTY	Balise auto-fermante comme dans 	`<!ELEMENT br EMTPY>`

Restriction sur les identificateurs (noms)

- chaque identificateur doit commencer par une lettre ou '_'
- ensuite lettres, chiffres, '_' , '-', '.', ':'

Comprendre/lire une DTD:

- pas trop difficile quand on a l'habitude
- Il faut partir de l'élément "root" (indiqué dans le DOCTYPE d'un fichier exemple)
- Ensuite voir comment est définit chaque sous-élément, et ainsi de suite.

Exemple 3-5: Une DTD pour un simple Address Book

```
 <!ELEMENT addressBook (person)+>

 <!ELEMENT person (name,email*)>

 <!ELEMENT name (family,given)>

 <!ELEMENT family (#PCDATA)>

 <!ELEMENT given (#PCDATA)>

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

- Questions:
  - quelle est la racine qu'on utilisera vraisemblablement ?
  - pour que le document soit valide il faut combien d'éléments "person" au moins ?
  - est-ce une DTD interne ou externe? (question piège)

Exemple 3-6: Exemple d'un arbre XML valide

```
<addressBook>

  <person>

    <name>  <family>Wallace</family> <given>Bob</given> </name>

    <email>bwallace@megacorp.com</email>

  </person>


  <person>

    <name>  <family>Tuttle</family> <given>Claire</given> </name>

    <email>ctuttle@megacorp.com</email>

  </person>

</addressBook>
```

Exemple 3-7: Exemple d'un arbre XML invalide

```
<addressBook>

    <address>Derrière le Salève</address>

    <person>

    <name>

      <family>Schneider</family> <firstName>Nina</firstName>

    </name>

    <email>nina@dks.com</email>

  </person>

  <name>

   <family> Muller </family> </name>

</addressBook>
```

Question:

- quelles sont les erreurs? (il y en a trois)

Exemple 3-8: Une DTD pour une recette

- Pour un exemple XML, voir l'exemple Une recette en XML

```
 <!ELEMENT list (recipe+)>

 <!ELEMENT recipe (author, recipe_name, meal, ingredients, directions)>

 <!ELEMENT author (#PCDATA)>
 <!ELEMENT recipe_name (#PCDATA)>
 <!ELEMENT meal (#PCDATA)>

 <!ELEMENT ingredients (item+)>
 <!ELEMENT item (#PCDATA)>
 <!ELEMENT directions (#PCDATA)>
```

- Voir: *http://tecfa.unige.ch/guides/xml/examples/simple-dtd/choco-chip.xml*

Exercice 1: Lecture d'une DTD

- L'élément recette contient combien d'éléments ?
- Lesquels de ces éléments sont optionels ?

3.4 Déclaration d'attributs

Attributs DTD simples (Voir la spec pour une définition "clean" !!):

<!ATTLIST target_tag attr_nom TypeAttribut TypeDef Defaut>

- Vouz avez un contrôle limité sur le type de contenus autorisés pour un attribut

	Types d'attributs (TypeAttribut)
ID	Permet de définir un identificateur unique pour un élément du document. Donc chaque id doit être différent ! Exemple d'usage: faire des tables de matière.
IDREF	Doit correspondre à un attribut "ID" dans un des éléments du document. Voir ci-dessus.
IDREFS	Doit correspondre à 1 ou plusieurs ID attributs (séparés par des blancs)
(A|B|C|..)	Liste énumérée de valeurs d'attributs à choix. Vous permet de contrôler un petit peu le contenu que les utilisateurs peuvent rentrer.
CDATA	"Character Data" - Contenu arbitraire, mais normalisé: espaces et fin de lignes convertis en un seul espace !
NMTOKEN	L'utilisateur peut rentrer un seul mot

	Explication de TypeDef
#IMPLIED	Attribut à option (l'utilisateur peut l'utiliser)
#REQUIRED	Attribut obligatoire (l'utilisateur doit rentrer une valeur)
#FIXED Value	Attribut avec valeur fixe (la valeur est déjà fixée dans la DTD)

Illustrations:

```
<!ATTLIST person prenom CDATA #REQUIRED>

<!ATTLIST person gender (male|female) #IMPLIED>

<!ATTLIST form method CDATA #FIXED "POST">

<!ATTLIST list type (bullets|ordered) "ordered">

<!ATTLIST sibling type (brother|sister) #REQUIRED>

<!ATTLIST person id ID #REQUIRED>
```

Attributs DTD multiples:

<!ATTLIST target_tag

attr1_nom TypeAttribut TypeDef Defaut

attr2_nom TypeAttribut TypeDef Defaut

...

>

Illustrations:

```
 <!ATTLIST person

       ident           ID                #REQUIRED

       gender          male|female)      #IMPLIED

       nom       CDATA            #REQUIRED

       prenom    CDATA            #REQUIRED

       relation  (brother|sister) #REQUIRED >


<!ATTLIST portable

     proprio   IDREF            #REQUIRED >
```

Exemple 3-9: Une DTD pour un Address Book plus complexe

[contenu du fichier ab.dtd]

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

 <!ELEMENT addressBook (person)+>

 <!ELEMENT person (name,email*)>

 <!ATTLIST person id ID #REQUIRED>

 <!ATTLIST person gender (male|female) #IMPLIED>

 <!ELEMENT name (#PCDATA|family|given)*>

 <!ELEMENT family (#PCDATA)>

 <!ELEMENT given (#PCDATA)>

 <!ELEMENT email (#PCDATA)>

 <!ELEMENT link EMPTY>

 <!ATTLIST link manager IDREF #IMPLIED subordinates IDREFS #IMPLIED>
```

Exemple d'un XML valide par rapport aux règles dans ab.dtd:

```
<!DOCTYPE addressBook SYSTEM "ab.dtd">

<addressBook>

  <person id="B.WALLACE" gender="male">

    <name>

      <family>Wallace</family> <given>Bob</given>

    </name>

    <email>bwallace@megacorp.com</email>

    <link manager="C.TUTTLE"/>

  </person>

  <person id="C.TUTTLE" gender="female">

    <name>

      <family>Tuttle</family> <given>Claire</given>

    </name>

    <email>ctuttle@megacorp.com</email>

    <link subordinates="B.WALLACE"/>

  </person>

</addressBook>
```

3.5 Attributs vs. Elements

- Il s'agit ici une grande FAQ sans réponse précise
- Ci-dessous quelques réflexions à pondérer ....

Il faut plutôt utiliser un élément

- lorsque l'ordre est important (l'ordre des attributs est au hasard)
- lorsqu'on veut réutiliser un élément plusieurs fois (avec le même parent)
- lorsqu'on veut (dans le futur) avoir des descendants / une structure interne
- pour représenter un type de données (objet) plutôt que son usage, autrement dit: une "chose" est un élément et ses propriétés sont des "attributs".
- lorsque XML sert comme markup pour un texte à publier (tout ce que le lecteur devra voir se trouvera dans un élément, tout ce qui est "meta" dans attributs)

Il faut plutôt utiliser un attribut

- lorsqu'on désire faire référence à un autre élément
  - <compagnon nom="lisa" genre="giraffe") fait référence à <animal cat="giraffe">
- pour indiquer l'usage/type/etc. d'un élément comme dans:
  <address usage="prof"> ... </address>
- lorsque vous voulez imposer des valeurs par défaut dans le DTD
- lorsque vous voulez un type de données (pas grand chose dans le DTD)

3.6 Déclaration d'Entités

- Une "entity" est un bout d'information stocké quelque part.
- Lors de l'utilisation d'un document, les entités sont remplacés par le contenu référencé

Seulement 5 entités sont prédéfinies (toutes pour les signes spéciaux)

- on vous rappelle à cette occasion qu'il faut toujours utiliser ces entités
  (même dans les URL qu'on utilise avec XHTML !)

Entity	Signe
&	&
<	<
>	>
"	"
'	'

Les autres entités doivent être définies par l'auteur de la DTD

- soit dans la DTD elle-même
- soit par une déclaration qui renvoie à un contenu défini ailleurs (un URI)

A. Entités générales

<!ENTITY nom_du_tag "contenu">

Illustrations:

```
<!ENTITY tecfaUnit "Unité de technologies de formation et apprentissage">

<!ENTITY tecfaDesc SYSTEM "http://tecfa.unige.ch/../tecfa_description.xml">

<!ENTITY pm "Patrick Mendelsohn">

<!ENTITY acirc  "&#194;">

<!ENTITY espace "&#160;">

<!ENTITY copyright "&#xA9;">

<!ENTITY explication SYSTEM "project1a.xml">
```

Référence à une entité générale (simple substitution)

```
<para> &pm;
 sort du ch&acirc;teau, s'<para>
```

va donner:

```
<para> Patrick Mendelsohn sort du ch&#194;teau</para>
```

Inclusion d'un contenu d'un autre fichier:

```
<para> blabla

<citation> &explication;
 </citation>

.... </para>
```

va donner:

```
<para> blabla

<citation> .... tout le contenu du fichier project1a.xml...
 </citation>

.... </para>
```

B. Entités "paramétriques"

- Aident à fabriquer des DTD complexes
- Exemple:

```
<!ENTITY % stamp
'

  id ID #IMPLIED

  creation-day NMTOKEN #IMPLIED

  .......

  mod-by NMTOKEN #IMPLIED

  version NMTOKEN #IMPLIED

  status (draft|final|obsolete) #IMPLIED

  approval (ok|not-ok|so-so) #IMPLIED

  main-author CDATA #IMPLIED

'

>
```

Usage: Les listes d'attributs ci-dessous contient tous les attributs définis dans l'entité %stamp;

```
<!ELEMENT main-goal (title, content, (after-thoughts)?, (teacher-comments)?)>

<!ATTLIST main %stamp;
 >

<!ELEMENT title (...)>

<!ATTLIST main %stamp;
 >
```

  - on évite donc de taper plusieurs fois les mêmes déclarations.

Exemple 3-10: Exemple DTD avec entités incluses dans une DTD

- *http://tecfa.unige.ch/lib/xml/dtd/ePBL11/*
- Il faut déclarer la DTD à inclure comme une ENTITIY externe et ensuite l'inclure.

Voici montré avec un exemple:

- Dans le cours staf-18 les étudiants uilisent la DTD "ePLpaper11.dtd" pour rédiger leur papier. Cette DTD inclut la DTD ibtwsh6.dtd et en utilise une partie:

```
<! ENTITY % foreign-dtd SYSTEM "ibtwsh6_ePBL.dtd" >

%foreign-dtd;
```

- ibtwsh6_ePBL.dtd est un sous-ensemble de XHTML qu'on utilise ici pour permettre la mise en forme à l'intérieur d'éléments XML "sémantiques". *%foreign-dtd;* va l'importer.
- Cette DTD a plusieurs "racines" définies.
  - %vert.model et %struct-model correspondent à une série de balises
  - Si on décide d'utiliser un modèle "très ouvert":

```
<!ELEMENT introduction %vert.model
;>
```

  - A Tecfa, on préfère *%struct.model* :

```
<!ELEMENT introduction %struct.model;
>

<!ELEMENT conclusion %struct.model;>
```

Activité:

- Essayez de voir ce que font *vert.model* et *struct.model* en examinant le fichier ibtwsh6_ePBL.dtd.

C. Annexe: La définition de XML

- XML (comme beaucoup d'autres grammaires en informatique) est décrit sous format EBNF (Extended Backus-Naur Form)
  - Comprendre EBNF est nécessaire pour bien comprendre la spécification de XML
    (mais ce n'est pas une nécessité absolue)

Exemple 3-11: Quelques éléments de XML à titre d'illustration

```
[1] document ::= prolog element Misc

[2] element  ::= EmptyElemTag | STag content ETag [WFC: Element Type Match]

                 [VC: Element Valid]
```
