Initiation à XML

Séquence 5

➤ ***Contenu***

Introduction

XML signifie **eXtensible Markup Language**. La première version de ce langage a vu le jour en 1998, il est donc beaucoup plus récent que html.

XML est le résultat de la coopération des membres du W3C dans l'objectif de définir **un formalisme permettant d'échanger facilement des documents sur le Web** en dépassant les limites imposées par HTML.

XML est considéré comme le langage qui sera utilisé dans l'avenir en remplacement de html. Créé par le w3c, xml est un langage permettant de créer des documents normalisés, possédant des caractéristiques indépendantes du logiciel ayant servi à les créer, et pouvant donc être échangés (comme les documents html, texte…).

1. Les origines de XML et un peu d'histoire

1A. Son ancêtre : SGML

XML est issu d'une norme existante : SGML (Standard Generalized Markup Language), née en 1986, une norme « très lourde » destinée à la GED (gestion électronique de documents), utilisée pour créer des documentations volumineuses (donc longues à créer) de type encyclopédique, dont le contenu, étant donné le temps de création, doit rester utilisable longtemps et doit donc rester indépendant de toute version de logiciel le manipulant (navigateur, éditeur …).

Dans SGML, tout document doit respecter une DTD (un « Document Type Definition »), sorte de « patron » décrivant la structure générique du document, et HyTime, le mécanisme de liens hypertexte entre les différents documents, est extrêmement complexe.

1B. La naissance de XML

En Juin 1996, le W3C se donne comme objectif de spécifier un langage alliant la richesse de SGML et la simplicité de HTML.

Actuellement, et depuis février 1998, on en est à la version 1.0 de la norme.

Le W3C décrit XML comme « une syntaxe de bas niveau permettant de représenter des informations de façon structurée ».

1C. Comparaison sommaire à HTML

HTML est actuellement très implanté sur le Web grâce, notamment, à sa facilité d'utilisation.

Cependant, il pose certains problèmes, comme par exemple :

– l'impossibilité de créer de nouvelles balises ;

– l'existence de balises et d'attributs spécifiques à certains navigateurs et non reconnus par les autres navigateurs ;

– le manque de rigueur syntaxique : en effet, malgré une balise non fermée ou manquante, le document sera affiché correctement avec la plupart des navigateurs, mais si un développeur veut exploiter le document en vue, par exemple, du stockage de son contenu dans une base de données, il a besoin de tester l'existence des balises afin de faire le tri entre ce qu'il doit stocker dans la base et ce qu'il doit laisser de côté. Les documents html sont donc difficilement exploitables par des traitements automatisés.

Le dépassement, par **xml**, des limites imposées par le HTML concerne les aspects suivants de HTML.

Description physique et structurelle mêlées : en effet, dans un document HTML, la mise en forme et le contenu du document peuvent être décrits dans le même fichier HTML (on peut décrire l'apparence d'une information en renseignant les attributs des balises qui contiennent cette information). Ceci n'est plus le cas avec XML.

XML sert uniquement à décrire les données.

En xml, la mise en forme d'un document en vue, par exemple, de sa présentation dans un navigateur, se fait uniquement par le biais de feuilles de style.

Conséquence : le document **xml** ne contient que des informations, structurées à l'aide de balises et d'attributs. Ceci permet, entre autres, de traiter de manière automatique le contenu d'un document **xml**, en vue du stockage des informations qu'il contient (par exemple dans une base de données). Il existe d'ailleurs un langage de requêtes spécifique, le langage **xquery** ou **xmlquery** (normalisé lui aussi), qui permet, à l'aide de requêtes, de récupérer les informations contenues dans un document **xml**.

Comme html, **xml** est un langage à balises. Dans html, on dispose d'un ensemble prédéfini de balises : HTML ne permet pas de créer de nouvelles balises. XML, lui, le permet, il est extensible. En effet, il est possible de créer ses propres variables, ses propres balises, et il est donc possible de créer des langages à balises : on pourrait, par exemple, recréer html à l'aide de xml. XML est donc un métalangage.

Il permet de créer des langages qui héritent de ses spécificités, appelés dialectes.

Quelques exemples de dialectes déjà très utilisés :

– MathML : description d'objets mathématiques ;

– SMIL : mise en œuvre de présentations multimédia ;

– SVG : langage graphique au format vectoriel.

Le langage XML pallie également aux défauts de html en étant très strict au niveau de la syntaxe. Tout navigateur refusera de visualiser un document mal formé (c'est à dire un document ne respectant pas les règles pour être un document bien formé, c'est-à-dire encore un document ayant des défauts de balises et/ou de structure).

1D. XML a plein d'acolytes

XML en lui-même est « juste » un langage de description de documents. Mais comme il fait l'objet d'une norme, il est au centre de tout un tas d'outils et de techniques qui font qu'un même document xml peut servir à tout un tas de choses.

XML étant destiné à de nombreuses utilisations, pour compléter la norme XML 1.0, le W3C propose des recommandations et modèles associées à ses différents contextes d'utilisation.

On en est actuellement aux alentours d'une cinquantaine de recommandations, dont voici quelques exemples. Nous nous initierons à certains de ces outils, pas à d'autres.

1D1. Les outils que je vous présente pour la culture

• La recommandation **Xlink** : elle décrit la manière d'ajouter des liens hypertextes à un document XML. Je vous conseille d'apprendre à utiliser des liens hypertextes à l'aide de **XLink**, c'est tout facile, il y a plein de sites proposant des tutoriels. Je ne le fais pas dans ce support, sinon, il sera trop long.

• La recommandation **DOM** (Data Object Model) : le Modèle Objet de Documents fournit une ensemble de standards d'objets pour représenter des documents HTML et surtout XML. Ce modèle spécifie des fonctions de manipulation des documents XML depuis un langage de programmation : analyse de structures, contrôles syntaxiques, validation d'un document par rapport à son «patron» (une DTD ou un schéma XSD, décrits plus bas).

À titre d'exemple, la fonction **getElementById** de javascript, qu'on a appris à utiliser voilà déjà un certain temps, est une fonction s'appuyant sur le Modèle Objet de Documents.

• Les recommandations **XPointer** et **Xfragments** : ces recommandations gèrent des pointeurs entre fragments de documents dans le but de construire des structures complexes.

• Le modèle **SAX** propose une **API** (une interface de programmation, c'est-à-dire des bibliothèques de fonctions, pour faire simple) de manipulation de flux XML spécifique à **Java**.

• **XML Schema** est une recommandation du W3C, succédant aux **DTD (Definition Type Document),** issues quant à elles de SGML. Ces deux outils servent à créer des fichiers de description de structure de document xml. Le mode d'utilisation est un peu le même que lorsqu'on crée une feuille de style css : on attache au document xml le fichier qui décrit la structure qu'il doit et peut avoir.

• **XMLQuery** : cette recommandation est un outil de manipulation des documents XML au sein des bases de données. Contrairement à d'autres propositions (comme XQL ou QXML), **XMLQuery** établit un pont entre le SQL et les langages de requêtes des SGBD objets.

• La recommandation **SOAP** (Simple Object Access Protocol). C'est un protocole d'échange de messages entre services web dont les formats de messages sont définis en XML.

La vague de recommandations la plus récente concerne tout ce qui tourne autour des architectures distribuées (avec SOAP comme protocole) comme par exemple :

– **XML-WSDL :** ce dialecte décrit la nature et les fonctions proposées par les composants des applications communiquant par échange de message et décrits à l'aide de schémas XML ;

– **UDDI :** définit un service d'annuaire facilitant la localisation distante d'un service web. Les requêtes sont formulées en XML ;

– **WSUI :** permet de définir en XML l'interface utilisateur éventuellement associée à chaque service.

Outre les recommandations du w3c, de nombreux langages de programmation proposent également au développeur tout un tas de fonctions pour manipuler les documents xml.

Après avoir fait ce cours, je vous conseille de regarder un peu du côté de php, pour voir comment il permet de manipuler les documents xml.

1D2. Les outils auxquels on va s'intéresser

Pour notre part, dans ce support, nous ferons simple. Voici les outils que l'on va tester :

– nous décrirons la structure que doit avoir un document xml à l'aide de l'outil **DTD**, même si le w3c a créé la recommandation **xml schema** plus récemment ;

– nous verrons les feuilles de style permettant de mettre en forme les documents xml : nous referons un peu de css, que vous connaissez déjà, qui est également utilisé pour les documents html, et je vous présenterai seulement les principes de **xsl**, plus spécialement conçu pour mettre en forme les documents xml.

2. Caractéristiques de XML

Note importante : comme on est à la fin du support de cours sur la programmation web, j'utilise une façon un peu plus technique de vous présenter les choses, car je pense que vous êtes maintenant en mesure de comprendre ce type d'explications.

2A. Caractéristiques techniques

XML est un langage de description de documents structurés en arborescences, permettant de décrire la structure logique de documents. La description est construite à partir d'une racine unique, en utilisant un système de balises utilisateurs (des « tags », les nœuds de l'arborescence). Les balises servent à marquer chaque élément de la structure décrite, ainsi que les relations entre les éléments de la structure. Les balises peuvent porter des attributs. Le nombre de nœuds est illimité.

Ca ressemble beaucoup au html, n'est-ce pas ? Les documents html ont aussi une structure arborescente, avec une racine unique (la balise <html>). Sauf que xml est extensible et possède d'autres caractéristiques intéressantes…

Extensibilité : XML est extensible et permet, au choix de l'utilisateur, de préférer soit une représentation « linéaire » donc pauvre et non extensible des données, soit une représentation arborescente, donc riche et extensible, par la création de balises adaptées.

Exemple :

Représentation pauvre	Représentation riche
<date> 11 Décembre 2002 </date>	<date> <jour>11</jour> <mois>Décembre</mois> <annee>2006</annee> </date>

Cette représentation « riche » peut s'apparenter à le décomposition atomique des données dans une base de données relationnelle.

Contrairement à la représentation « linéaire » de la date, elle permet, par exemple, de ne faire apparaître qu'un seul des éléments de la date, de modifier aisément un seul des éléments de la date…

Modularité et réutilisabilité : XML, au moyen de descriptions dans des documents externes, permet la réutilisation de structures arborescentes préalablement définies comme par exemple la date ci-dessus. Cela revient à dire que XML permet de créer des types de données structurés et réutilisables.

Contrôle de validité : XML a hérité de SGML la possibilité (mais non l'obligation) de doter les documents XML d'une DTD (Document Type Definition) afin de vérifier que leur structure obéit aux règles énoncées dans la DTD.

Même s'il possède une DTD, un document peut être distribué sans sa DTD, et sans référence explicite à cette DTD.

Deux niveaux de contrôle de validité sont proposés par la norme : ces deux niveaux de contrôle servent à la mise en œuvre des parsers XML (les analyseurs de code source XML).

Premier niveau : un document est dit **bien formé** (well formed) s'il obéit aux règles syntaxiques du langage XML,

Second niveau : un document est dit **valide et bien formé** s'il obéit à une structure type définie explicitement dans une DTD ou un schéma.

Intéropérabilité : la normalisation ouvre la possibilité d'échanges de données entre différents systèmes hétérogènes.

Actuellement, XML étant pressenti comme le futur outil de l'interopérabilité entre environnements hétérogènes, tous les environnements de développements proposent des outils pour développer des applications exploitant XML : éditeurs, parsers, générateurs de DTD et de schémas à partir de structure de données ou de classes (C++, C#, java), utilisation de XML pour décrire des métadonnées (WebDAV, RDF…).

Les éditeurs de SGBD du marché (DB2, Oracle, SQLServer…) permettent de générer des résultats de requêtes sous forme de documents XML en s'appuyant sur XML Query.

Sécurisabilité : les documents XML étant des documents texte, on peut, pour sécuriser les échanges, leur appliquer tous principes de sécurisation applicables aux fichiers textes (notamment le cryptage).

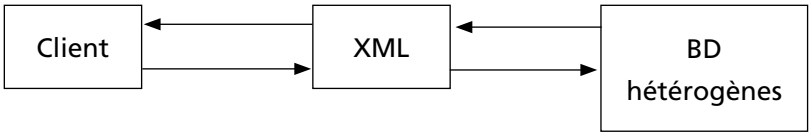
Compressibilité : les documents XML se compressent très bien, leur contenu étant souvent redondant (puisque décrivant des structures de données).

2B. À quoi peuvent servir les documents XML ?

En lui-même, XML est simple : c'est un langage permettant, au moyens de balises que l'on crée, de décrire le contenu d'un document textuel.

Les utilisations possibles d'XML, par contre, sont multiples : outre la structuration et la présentation de pages web, on peut citer, entre autres, l'exploitation native des structures XML en passant par les serveurs d'applications, les systèmes d'échanges de données entre applications (EAI), les services WEB, les systèmes de stockage et de manipulation de données et peuvent être très complexes car XML est déjà au centre d'un important fourmillement de nouvelles technologies qui se dirigent vers le Web sémantique et les entrepôts de données issues de systèmes hétérogènes.

XML est notamment en passe de devenir l'interface dans les architectures trois-tiers :



3. Structure d'un document XML

Nous allons d'abord commencer, dans ce paragraphe, par nous intéresser à la structure d'un document XML « bien formé », c'est à dire respectant les règles du XML.

Nous nous intéresserons plus loin aux documents xml « valides et bien formés », c'est-à-dire utilisant une DTD ou un schéma.

Un document bien formé se compose :

– d'un prologue (obligatoire) ;

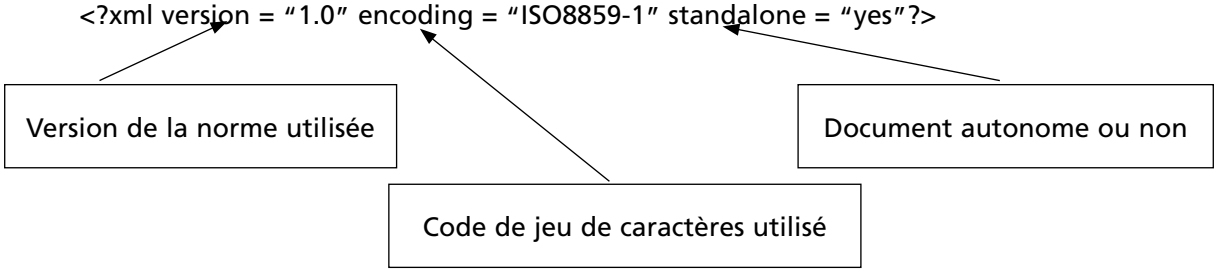
– d'instructions à usage de traitements probables (facultatives) ;

– d'un arbre d'éléments (obligatoire).

3A. Le prologue

Il indique par exemple les informations suivantes, obligatoirement encadrées par <?….?> :

<?xml version = "1.0" encoding = "ISO8859-1" standalone = "yes"?>



Remarques

Standalone *= "**yes**" signifie que le document xml n'utilise pas de fichiers externes. Si le document xml a recours à des fichiers externes (comme une DTD, par exemple), il faut écrire standalone = "no".*

Le jeu de caractères ISO8859-1 *correspond aux caractères utilisés en français.*

On peut compléter le prologue avec d'autres instructions facultatives de traitements, à destination d'applications particulières. Ces instructions sont toujours encadrées par <?…?>.

3B. Instructions complémentaires

C'est dans cette partie que l'on indique, par exemple, à quelle DTD se réfère le document, dans le cas d'utilisation d'une DTD externe.

Exemple : <!DOCTYPE rapport SYSTEM "rapport.dtd">

3C. L'arbre d'éléments

C'est là que l'on décrit les éléments décrivant le document. Cette partie est détaillée dans le paragraphe ci-dessous.

3C1. Règles de description de l'arbre d'éléments

Chaque élément d'un document XML se compose d'une balise d'ouverture, d'un contenu et d'une balise de fermeture.

Exemple : <nom> contenu de l'élément </nom>

Si dans un document, on veut laisser un élément vide, on peut simplifier l'écriture de cet élément vide : <nom/>

Les éléments peuvent contenir des sous-éléments. Dans ce cas, la seule règle à respecter est la suivante : tout élément fils est entièrement inclus dans son père. Il faut en fait voir les balises comme des parenthèses et dans ce cas, un document XML doit être complètement parenthésé.

Tout document XML contient une balise racine contenant toutes les autres balises (qui elles-mêmes sont la racine des balises qu'elles contiennent).

On peut mettre des commentaires : <!- - Ceci est un commentaire - - >

Remarque

Le plus petit document xml bien formé est constitué d'un élément vide.

Exemple : <monDocVide/>

3C2. Un exemple de document xml

Supposons que l'on veuille créer un document XML sur des étudiants du BTS IG, avec des renseignements sur chaque option.

Pour chaque étudiant, on veut afficher : son nom, son prénom, l'option à laquelle il est inscrit et les deux matières les plus prépondérantes de son option, ainsi que le professeur principal de l'option.

Voici le document xml concernant 3 étudiants… Je l'ai appelé **etudiants.xml**

```
<?xml version = "1.0" encoding = "ISO8859-1" standalone = "yes"?>
<btsIG>
    <etudiant>
    <nom>Tartempion</nom>
    <prenom>Gudule</prenom>
    <option>
          <nomoption>Réseau</nomoption>
          <matieresprepondérantes>
```

```
                <matiere1> Réseau </matiere1>
                <matiere2> Système </matiere2>
          </matieresprepondérantes>
          <profPrincipal> Madame Truc </profPrincipal >
    </option>
    </etudiant>

    <etudiant>
    <nom> Shmolldu </nom>
    <prenom>Alphonse </prenom>
    <option>
          <nomoption>Développeur</nomoption>
          <matieresprepondérantes>
                <matiere1> Algorithmique et développement</matiere1>
                <matiere2> Base de données </matiere2>
          </matieresprepondérantes>
          < profPrincipal > Monsieur Bidule </profPrincipal >
    </option>
    </etudiant>

    <etudiant>
    <nom> Bob </nom>
    <prenom> Machin </prenom>
    <option>
          <nomoption>Réseau</nomoption>
          <matieresprepondérantes>
                <matiere1> Réseau </matiere1>
                <matiere2> Système </matiere2>
          </matieresprepondérantes>
          <profPrincipal> Madame Truc </profPrincipal >
    </option>
    </etudiant>
<-- Et ainsi de suite pour chaque étudiant -->
</btsIG >
```

Vous constatez que les informations communes à chaque étudiant sont répétées pour chacun d'eux, ce qui donne un résultat assez lourd, même s'il est syntaxiquement correct.

Pour limiter la redondance d'information, on a la possibilité d'avoir recours à la création d'attributs.

3D. Affectation d'attributs aux éléments

3D1. Syntaxe générale

On peut doter un élément d'un ou plusieurs attributs, suivant la syntaxe suivante :

```
<NomDeL'élément attribut1 = "valeur1" attribut2 = "valeur2"…attributn =
"valeurn">
…..
<-- Description de l'élément -->
……
</NomDeL'élément>  <-- Fin de la description -->
```

Doter un élément d'attributs sert à rendre un document plus lisible et à éviter la redondance des informations.

Exemple : <rapport langue = "FR" date-modif = "08.12.89" diffusion = "restreinte"/>

3D2. Reprise de l'exemple des étudiants, avec attributs

Exemple : nous allons reprendre l'exemple du paragraphe précédent, qui n'utilisait pas d'attributs, et le transformer de manière à ce qu'il utilise des attributs. J'ai appelé le fichier correspondant **etudiantsAvecAttributs.xml.**

```
<?xml version = "1.0" encoding = "ISO8859-1"?>
<btsIG>
  <options> <!-- On commence par décrire les options -->
     <option nomoption = "Réseau">
    <matieresprepondérantes>
          <matiere1> Réseau </matiere1>
          <matiere2> Système </matiere2>
    </matieresprepondérantes>
    <profPrincipal > Madame Truc </profPrincipal >
     </option>

     <option nomoption = "Développeur">
         <matieresprepondérantes>
          <matiere1> Algorithmique et développement </matiere1>
          <matiere2> Base de données </matiere2>
         </matieresprepondérantes>
         <profPrincipal> Monsieur Bidule </profPrincipal>
     </option>
  </options>
<!-- Maintenant, on décrit la liste des étudiants en "appelant" simplement la
bonne option, -->
<!-- décrite ci dessus -->
```

```
    <etudiant nom = "Tartempion" prenom = "Gudule" nomoption ="Réseau"/>
    <etudiant nom = "Shmolldu" prenom = "Alphonse" nomoption ="Développeur"/>
    <etudiant nom = "Truc" prenom = "Machin" nomoption ="Réseau"/>
<!-- Et ainsi de suite pour chaque étudiant -->
</btsIG>
```

Ces deux documents XML (celui ci-dessus et celui du paragraphe précédent) sont équivalents, dans la mesure où ils contiennent la même quantité d'information, mais celui avec attributs est plus lisible et moins redondant que celui sans les attributs.

⚠ ***Le fait que ces deux documents soient équivalents en terme de quantité d'information n'implique pas qu'on puisse leur appliquer les mêmes traitements (c'est-à-dire que si on décide d'appliquer des requêtes écrites en xquery, pour par exemple, afficher la liste des étudiants par option, la requête n'aura pas la même syntaxe selon que l'on veuille exploiter le document ci-dessus ou celui du paragraphe précédent).***

3E. Faisons quelques tests sur ces exemples…

3E1. Affichons un document xml dans notre navigateur

Ouvrez donc **etudiants.xml** dans votre navigateur… rien de chouette, ça ne donne rien de chouette : en fait, cela nous affiche le code source xml de notre document… C'est que, comme je vous l'ai déjà indiqué plus haut, si on veut permettre la visualisation du contenu d'un document xml dans un navigateur, il faut absolument lui adjoindre une feuille de style…

Voici un extrait de ce que je vois dans mon navigateur :

```
<?xml version="1.0" encoding="ISO8859-1" standalone="yes" ?>
- <btsIG>
- <etudiant>
   <nom>Tartempion</nom>
   <prenom>Gudule</prenom>
- <option>
```

Vous avez remarqué ces petits traits devant certaines balises ? Si vous positionnez votre curseur de souris dessus, il change de forme, et si vous cliquez sur ces traits, ils ferment la partie d'arborescence dont la balise qu'ils précèdent est la racine. Une portion d'arborescence une fois fermée, c'est un signe + qui remplace le signe -.

3E2 Visualiser les données, mais pas la structure

Afin de visualiser les documents, mais sans leurs attributs, rajoutez, entre la ligne **<?xml version = "1.0" encoding = "ISO8859-1" standalone = "yes"?>** et la ligne <btsIG>, la ligne suivante :

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

Dans la ligne **<?xml version = "1.0" encoding = "ISO8859-1" standalone = "yes"?>**, remplacez **yes** par **no**, puisque maintenant, notre document xml utilise une ressource externe.

Le fichier monStyle.css n'existe pas et nous n'allons pas le créer pour le moment. Rien que le fait que l'on rajoute cette ligne dans le fichier xml rend possible sa visualisation dans notre navigateur.

Voilà, lorsque vous affichez **etudiants.xml** dans votre navigateur, vous voyez quelque chose de très moche, car pas mis en forme du tout, puisque vous voyez ça :

Tartempion Gudule Réseau Réseau Système Madame Truc Shmolldu Alphonse Développeur Algorithmique et développement Base de données Monsieur Bidule Bob Machin Réseau Réseau Système Madame Truc

Mais au moins, ici, seules les informations (c'est-à-dire le contenu du document xml), sont affichées, et pas la structure.

3E3. Réactions du navigateur en cas d'erreurs de syntaxe

Prenez maintenant le fichier **etudiantsAvecAttributs.xml**. Affichez-le dans votre navigateur.

Bon, comme on ne lui a pas associé de feuille de style, c'est son code source qui s'affiche, mais ce n'est pas grave, ce qu'on veut, c'est provoquer des erreurs pour tester les réactions de notre navigateur.

Première erreur : elle porte sur les commentaires.

Remplacez la ligne :

<options> <!-- On commence par décrire les options -->

par :

```
<options> <-- On commence par décrire les options -->
```

Enregistrez, puis actualisez l'affichage de votre document xml dans votre navigateur…

Aaaahh ! Notre premier message d'erreur :

```
La page XML ne peut pas être affichée
Impossible d'afficher l'entrée XML en utilisant la feuille de style XSL.
Corrigez l'erreur, puis cliquez sur le bouton Actualiser ou réessayez ultérieurement.


Nom commencé avec un caractère non valide. Erreur de traitement de la ressource file:///C:/fred/ XMLXquery/es...
  <options> <-- On commence par décrire les options -->
-------------^
```

Si vous provoquez la même erreur dans le commentaire figurant dans le fichier etudiants.xml, le navigateur nous parle d'une feuille de style css, ce qui est normal puisqu'on lui a déclaré qu'on utilisait une feuille de style css.

Note : *apparemment, lorsqu'on n'associe aucune feuille de style à un document xml, le navigateur lui en associe une par défaut, et c'est une feuille de style xsl. Ce qui me laisse à penser cela, c'est la partie du message d'erreur qui dit…*

```
Impossible d'afficher l'entrée XML en utilisant la feuille de style XSL.
```

Mais ce qui nous intéresse ici, dans ce message d'erreur, c'est cette partie :

```
Nom commencé avec un caractère non valide. Erreur de traitement de la ressource file:///C:/fred/ XMLXquery/es...
  <options> <-- On commence par décrire les options -->
-------------^
```

Les erreurs sont détectées par le parser xml intégré au navigateur. Le navigateur ne reconnaît pas notre commentaire, puisqu'on lui a enlevé son point d'exclamation. Il refuse alors d'afficher le fichier xml, et nous signale l'emplacement de l'erreur. Je corrige… et j'essaie de faire une autre erreur.

Deuxième erreur : j'essaie de changer les valeurs de l'attribut standalone dans les deux fichiers xml.

Dans le fichier **etudiants.xml**, qui utilise une feuille de style externe, je remplace :

```
standalone = "no"
```

par :

```
standalone = "yes"
```

Je teste. Ca ne change rien lorsqu'on visualise le document dans le navigateur.

Dans le fichier **etudiantsAvecAttributs.xml**, qui n'utilise aucune ressource externe, je remplace :

```
<?xml version = "1.0" encoding = "ISO8859-1" ?>
```

par :

```
<?xml version = "1.0" encoding = "ISO8859-1" standalone = "no"?>
```

Je teste. Ca ne change rien non plus lors de la visualisation.

Ce que je **remarque**, par contre, c'est que même si je mets des espaces, comme par exemple dans la ligne ci-dessus, de part et d'autre des signes =, lors de la visualisation, ces espaces ont été supprimés. C'est probablement le parser xml qui fait ça (**rappel : un parser,** c'est un outil logiciel qui analyse la syntaxe des fichiers. Un parser xml analyse la syntaxe des documents xml. Les navigateurs possèdent un parser xml et le sollicitent lorsqu'on viusalise un document xml).

Troisième erreur : j'enlève une balise fermante.

Par exemple, dans **etudiants.xml**, je remplace :

```
<nom>Tartempion</nom>
```

par :

```
<nom>Tartempion
```

Et voilà ce que me répond le navigateur lorsque je demande à visulaiser etudiants.xml.

```
La balise de fin etudiant ne correspond pas à la balise de début nom. Erreur
de traitement de la ressource file:///C:/fred/...
 </etudiant>
---^
```

Donc, effectivement, le contrôle des balises fermantes est beaucoup plus sévère qu'en html.

Je vais remettre la balise fermante </**nom**> à ce brave Tartempion, et ça va me donner l'occasion de faire encore une nouvelle erreur…

Quatrième erreur : un espace dans la balise.

Je remplace :

```
<nom>Tartempion
```

Par :

```
<nom>Tartempion</ nom>
```

J'ai mis un espace entre </ et **nom**>.

Et voilà le message d'erreur qui s'affiche :

```
Aucun espace blanc n'est autorisé à cet emplacement. Erreur de traitement de
la ressource file:///C:/fred/dess/ProjetEOL/XM...
 <nom>Tartempion</ nom>
------------------^
```

Eh ben dis donc !! C'est une surveillance sévère, hein !!

Même message d'erreur si je mets l'espace entre < et /**nom**>, mais pas si je mets l'espace entre **</nom** et >.

J'ai droit aussi au même message si j'insère un espace entre < et **nom**>.

Cinquième erreur : non respect de la casse.

Je remplace :

```
<nom>Tartempion</nom>
```

par :

```
<nom>Tartempion</NOM>
```

En html, on peut indifféremment écrire une des balises en minuscule et l'autre en majuscule, ça marche… mais pas en xml… la preuve…

```
La balise de fin NOM ne correspond pas à la balise de début nom. Erreur de
traitement de la ressource file:///C:/fred/dess/...
 <nom>Tartempion</NOM>
------------------^
```

Par contre, je peux écrire ça :

```
<NOM>Tartempion</NOM>
```

Donc, un couple de balises doit être écrit dans la même casse.

Bon, on a provoqué quelques erreurs intéressantes, cela nous permettra de ne pas être perdu si on rencontre des messages d'erreur du même type.

Passons maintenant à la mise en forme des documents xml à l'aide de feuilles de style.

4. Mise en forme de documents XML à l'aide de feuilles de style

Contrairement à un document HTML, si on n'associe pas une feuille de style à un document XML, c'est son code source qui s'affiche dans le navigateur lorsqu'on demande sa visualisation.

Pour n'afficher que les informations, et pas la structure, il faut obligatoirement lui adjoindre un fichier appelé « feuille de style », décrivant sa mise en forme.

Les feuilles de style peuvent être créées à l'aide de langages spécialement mis au point pour créer les feuilles de style.

Il existe deux principaux langages préconisés par le W3C pour créer des feuilles de style :

– CSS pour Cascading Style Sheet, le premier à avoir vu le jour, un langage assez simple mais limité, auquel vous vous êtes initiés récemment, il me semble, non ?

– XSL-FO, pour Extended Style Sheet, un langage beaucoup plus élaboré permettant d'effectuer de véritables traitements de mise en forme des données.

Exemple : soit le document XML **mapage.xml**, dont voici le code source :

```
<?xml version = "1.0" encoding = "iso8859-1" ?>
<Doc>
Oh! Là, une belle page Web!
</Doc>
```

Si j'ouvre **mapage.xml** dans Internet Explorer ou dans NetScape, voilà exactement ce qui s'affiche :

Maintenant, si je complète **mapage.xml** en lui adjoignant une feuille de style comme suit :

```
<?xml version="1.0" encoding="iso8859-1"?>
<?xml-stylesheet type="text/css" href="monStyle.css"?>
<Doc>
Oh! Là, une belle page Web!
</Doc>
```

Le texte de la page s'affiche, même si le fichier **monStyle.css** n'existe pas, ce qui est le cas pour le moment.

Lorsque j'ouvre **mapage.xml** depuis un navigateur de dernière génération, voilà ce qui s'affiche :

Si, par contre, j'essaie de déclarer que j'utilise une feuille de style xsl qui n'existe pas, comme ça…

```
<?xml version="1.0" encoding="iso8859-1"?>
<?xml-stylesheet type="text/xsl" href="monStyle.xsl"?>
<Doc>
Oh! Là, une belle page Web!
</Doc>
```

La page ne s'affiche pas, mais à la place, j'ai le message d'erreur suivant :

```
Le système ne trouve pas la ressource spécifiée. Erreur de traitement de la ressource
```

Si maintenant, je crée un fichier monStyle.xsl vide (par exemple à l'aide du bloc notes), ça ne s'affiche pas plus, mais j'ai un autre message d'erreur :

```
Le document XML doit contenir un élément de niveau supérieur. Erreur de traitement de la ressource
```

Bon, visiblement, les navigateurs sont plus sévères dès qu'il s'agit de feuilles de style xsl…

4A. Avec css

Essayons d'abord de nous faire une petite feuille de style telle que le texte **Oh! Là, une belle page Web !** s'affiche en rouge.

Allez, à vue de nez et par rapport à ce qu'on a fait dans le chapitre dernier, je crée un fichier texte que j'appelle **monStyle.css** et dans lequel j'écris :

```
Doc {color = red;}
```

J'enregistre ce fichier **monStyle.css**, et dans **mapage.xml**, j'indique que je l'utilise.

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

J'affiche **mapage.xml** dans mon navigateur… Bingo !! Le texte s'affiche en rouge.

Donc, j'en déduis qu'on peut utiliser css très facilement avec nos balises xml, comme s'il s'agissait de balises html… du coup, ça va être très facile.

On va d'ailleurs mettre en forme notre document xml **etudiants.xml**, en modifiant **monStyle.css** pour qu'il permette un affichage mis en forme de notre document.

4A1. Mise en forme du document etudiants.xml

Dans **monStyle.css,** rajoutez :

```
btsIG {color = green;}
```

Testez en visualisant **etudiants.xml.** Vous constatez que la couleur verte s'applique à tout le document.

Rajoutez :

```
nom {color = pink; font-size = 24;}
```

Vous constatez que toutes les informations du document sont vertes, sauf celles pour les balises nom, pour lesquelles on a spécifié une autre mise en forme.

Bon, maintenant, moi, j'aimerais bien aller à la ligne pour chaque information.

Lorsqu'on a mis en forme des documents html à l'aide d'une feuille de style css, on ne s'est pas préoccupé des passages à la ligne, que l'on a mis dans le document html, et pas dans le fichier css, ce qui n'est pas possible en xml, car même un passage à la ligne, c'est une mise en forme.

Il faut dons indiquer les passages à la ligne dans le fichier css, à l'aide de la règle css :

```
display: block;
```

Chaque fois que l'on indique cette règle pour une balise, alors, on aura, lors de la visualisation, un passage à la ligne après affichage d'une information relevant de cette balise.

Voici un exemple de style très sommaire utilisant le passage à la ligne, contenu dans le fichier **monStyle.css** :

```
Doc {color = red;}
btsIG {color = green;}
nom {display: block; color = pink; font-size = 24;}
prenom {}
nomoption {display: block;}
matiere1, matiere2 {display: block;}
profPrincipal {display: block;}
```

Et si on veut, par exemple, que le texte soit indenté (c'est-à-dire décalé vers la droite par rapport au reste du texte), il faut utiliser la règle css **text-indent**.

Par exemple, si je veux décaler le nom des deux matières prépondérantes vers la droite, je vais remplacer la règle :

```
matiere1, matiere2 {display: block;}
```

par :

```
matiere1, matiere2 {display: block; text-indent = 20;}
```

Et si je veux que mes deux matières prépondérantes apparaissent sous forme d'une liste à puces, avec des puces carrées, je vais écrire :

```
matiere1, matiere2 { text-indent = 20; display: list-item; list-style-type:
square; }
```

Je vous laisse continuer les essais de mise en forme en vous servant des quelques exemples figurant ci-dessus, et en vous reportant au chapitre précédent pour voir de quelles règles vous pouvez vous servir.

Moi, ce que je voudrais maintenant, c'est qu'on travaille sur le document **etudiantsAvecAttributs.xml.**

4A2. Mise en forme du document etudiantsAvecAttributs.xml

D'abord, il faut commencer par associer notre feuille de style **monStyle.css** à notre document **etudiantsAvecAttributs.xml.**

On rajoute une ligne dans notre document, de manière à ce qu'il commence comme ça :

```
<?xml version = "1.0" encoding = "utf-8" standalone = "no"?>
<?xml-stylesheet type = "text/css" href = "monStyle.css"?>
<btsIG>
…
```

Tiens, au passage, vous avez remarqué, j'ai changé de jeu de caractères, j'ai choisi **utf-8**, que vous connaissez pour l'avoir utilisé dans **mySql**.

Allez, on visualise **etudiantsAvecAttributs.xml** dans notre navigateur.

Oh, grûût alors !! J'ai un message d'erreur qui me dit que :

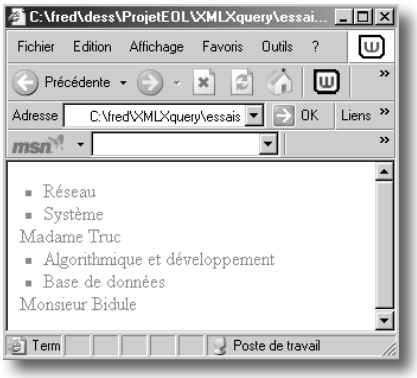
```
Un caractère incorrect a été trouvé dans un contenu de texte. Erreur de trai-
tement de la ressource file:///C:/fred/dess/Pro...
  <options> <!-- On commence par d
```

Cette erreur n'a pas l'air d'avoir quoi que ce soit à voir avec l'ajout de ma feuille de style… On dirait que c'est la lettre é se trouvant dans mon commentaire qui pose problème…

Bon, je remplace à nouveau **utf-8** par **ISO8859-1** pour voir si ça vient de là…

Oui, ça vient de là… Les accents ne sont pas reconnus lorsque le jeu de caractère est **utf-8**, donc, repassons en **ISO8859-1**.

Je visualise à nouveau le document dans mon navigateur. Ah, on n'a qu'un bout de notre document qui s'affiche…



Bon, ce qu'on peut d'ores et déjà constater, c'est qu'on ne peut pas appliquer la même feuille de style quand on factorise les attributs.

La partie qui s'affiche est celle pour laquelle il y a des balises dont la mise en forme a été décrite dans la feuille de style.

Je m'explique. Les seules balises effectivement présentes dans le document **etudiantsAvecAttributs.xml** sont les balises <btsIG>, <options>, <option>, <matieresprepondérantes>, <matiere1>, <matiere2>, <profPrincipal > et leurs balises fermantes.

Le nom, le prénom et l'option de chaque étudiant sont des attributs de la balise <**etudiant**>.

Parallèlement à cela, notre fichier css décrit la mise en forme des balises du document xml, et comme le nom, le prénom et l'option choisie par l'étudiant ne sont pas des balises, mais des attributs de la balise <**etudiant**>, ils ne s'affichent donc pas.

Ce que je voudrais, c'est obtenir, pour le document **etudiantsAvecAttributs.xml,** la même mise en forme que pour le document **etudiants.xml,** sans rien changer au code xml du fichier **etudiantsAvecAttributs.xml.**

Ce qu'il faut donc, c'est trouver comment, avec css, on demande l'affichage des valeurs des attributs des balises.

Pour le moment, ce qu'on a vu en css ne nous permet pas de faire ça.

Je cherche sur le net et je découvre qu'il existe, en css, une fonction **attr()** qui permet de faire ça, mais qu'apparemment, cela ne fonctionne pas sous internet explorer.

On va essayer quand-même.

Si ça ne fonctionne pas sous internet explorer, il faudrait alors qu'on utilise xsl, le langage de feuilles de style plus évolué.

Voici l'exemple, trouvé sur le net, sur lequel je m'appuie.

On peut utiliser la fonction attr().

```
Exemple en CSS2 pour une balise <Chap numero="1" id="XXX" page="32">, la CSS2
suivante devrait afficher "Chapitre : 1, ID XXX, page 32»
Chap:before {
content: "Chapitre : " attr(numero) ", ID [" attr(id) "], page " attr(page);
font-size: 16pt;
text-align: center;
display: block;}
```

Donc moi, pour faire simple, dans mon fichier css, j'écris :

```
Etudiant:before { content: "nom: " attr(nom) ; display: block; }
```

Juste pour voir si je peux au moins afficher le nom de chaque étudiant… Non, le nom ne s'affiche pas.

J'ai ça dans mon navigateur :

```
Réseau Système Madame Truc Algorithmique et développement Base de données
Monsieur Bidule
```

C'est-à-dire que j'ai encore et toujours les balises qui s'affichent, mais je ne parviens pas à afficher les valeurs des attributs.

En outre, avec css, il faudrait aussi trouver le moyen de masquer l'affichage des balises, pour n'afficher que les valeurs des attributs, et ça, ça se fait à l'aide de la règle **display :none**; que je vous laisse tester.

Bon, c'est vraiment bête qu'on ne puisse pas afficher les valeurs des attributs avec css lorsqu'on utilise internet explorer…

Il faudrait donc faire ça avec xsl.

Je n'avais pas prévu de vous présenter les feuilles de style xsl, car cela alourdit l'initiation qui est déjà bien dodue. Je vous fais, ci-dessous, une présentation théorique très rapide de xsl, et j'invite ceux que cela intéresse à s'y mettre en utilisant le net.

4B. Styles avec XST

Tout d'abord, xsl permet de faire beaucoup plus que css ne le permet.

De même que l'on peut utiliser css avec html et xml, on peut utiliser xsl avec html et xml.

Cependant, css a plutôt été conçu pour accompagner html, et xsl a été conçu pour compléter xml, afin de mettre en forme les documents xml.

XSL (eXtensible StyleSheet Language) est un langage, recommandé par le W3C, permettant de définir des feuilles de style pour les documents XML au même titre que les CSS (Cascading StyleSheets) pour le langage HTML.

XSL est lui-même défini avec le formalisme XML, cela signifie **qu'une feuille de style XSL est un document XML bien formé.**

Toutefois, et contrairement aux CSS, **XSL permet aussi de retraiter un document XML afin d'en modifier totalement sa structure.**

Il peut ainsi générer d'autres types de documents (Pdf, HTML, TxT, ...) ou bien un fichier XML de structure différente (vous voyez, c'est bien de xsl dont on aurait besoin pour mettre en forme notre document **etudiantsAvecAttributs**).

Les feuilles de style xsl sont, comme les fichiers css, dans des fichiers séparés.

Le XSL permet :

– d'afficher des informations ;

– de sélectionner une partie des informations ;

– de trier les éléments XML ;

– de filtrer les éléments XML ;

– de choisir des éléments XML ;

– de retenir que les éléments désirés à travers des tests conditionnels…

XSL se subdivise en deux langages :

– XSL-FO (eXtensible Stylesheet formatting : c'est ce qui est appelé XSL par abus de langage. C'est un langage permettant de décrire la mise en forme des documents XML afin d'en faire une présentation visuelle ;

– **XSLT** (eXtensible Stylesheet Transformation) : c'est un langage qui transforme un document XML en un format reconnu par un navigateur (Pdf, HTML, TxT, ou un autre document xml).

Voilà pour la petite présentation de ce qu'est xsl. Une chose est sûre, c'est que, au même titre que xml supplantera html dans l'avenir, xsl supplantera css.

C'est la même chose dans le paragraphe ci-dessous, avec les outils de description de la structure des documents xml : les schémas xml supplanteront les DTD dans l'avenir.

Et comme je vous présente les DTD et pas les schémas xml, de là à dire que je ne vous présente que des ringardises… il n'y a qu'un pas. Mais, pour le moment les feuilles de style **CSS** et les **DTD** présentées ci-dessous en encore le vent en poupe, et comme ces deux outils sont plus simples à appréhender que **XSL** (pour les feuilles de style) et **XMLSchema** (pour la description de la structure des documents xml), j'ai préféré aller au plus simple.

5. Outils de description de la structure des documents : DTD et schémas

XML impose une syntaxe définie par le W3C mais n'impose aucune restriction au niveau de la structure des documents.

Les **schémas xml** et les **DTD** (Document Type Definition) permettent de définir une telle structure.

Lorsqu'on utilise une DTD ou un schéma pour décrire la structure générique d'un document, le document XML, s'il respecte la structure décrite dans le **schéma** ou la **DTD** qu'on lui associe, est alors :

– bien formé (parce qu'il respecte la syntaxe xml) ;

– valide (parce qu'il possède la structure décrite dans sa DTD ou son schéma).

Une DTD ou un schéma peut être défini à l'intérieur du document xml ou séparément, dans un fichier externe, ce qui permet d'appliquer les mêmes contraintes de structure à plusieurs documents xml.

Les DTD et schémas ont le même objectif de validation de la structure d'un document XML.

Je vous fais ci-dessous une petite présentation rapide de ces deux recommandations du w3c.

Il n'existe aucune obligation, pour les concepteurs de documents xml, d'associer une DTD ou un schéma à tout document. Alors à quoi est-ce que ça sert ?

D'abord, ça sert à se créer un référentiel en ce qui concerne la structure des documents à produire. En particulier quand les données sont conservées sur de longues périodes et manipulées ou modifiées par des personnes différentes, une DTD est un document de base important pour les normes assurant l'exactitude et l'homogénéité des données.

Ensuite, cela sert aux développeurs qui écrivent des traitements sur des documents xml : ils vont s'appuyer sur la DTD à laquelle se réfèrent les documents xml pour écrire les traitements d'exploitation de ces documents, et ce, sans risque de perte d'information.

5A. DTD

La DTD d'un document XML contient la liste des balises et des attributs valides au sein du document, ainsi que leur logique d'imbrication.

Les éléments d'une DTD servent à définir les balises du langage que l'on veut créer. Chaque balise du langage devra faire l'objet d'une définition d'élément.

Exemple de DTD très simple (je l'ai appelée **DTDExemple.dtd)** :

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!-- début de la DTD -->
<!ELEMENT exemple (#PCDATA)>
<!-- fin de la DTD -->
```

Cette DTD indique que les documents xml auxquels on l'associe possèdent, pour toute balise, une balise s'appelant **exemple**.

#PCDATA indique que l'élément (la balise) **exemple** est une donnée de type chaîne de caractères (PCDATA signifie Parsed Character DATA, chaîne de caractères analysée). Ceci n'est pas toujours le cas : une donnée peut-être d'un autre type, ou bien l'élément peut être composé d'autres éléments.

Voici le code source d'un document XML valide et bien formé, je l'ai appelé **exempleDTD.xml** :

```
<?xml version = "1.0" encoding = "ISO-8859-1" standalone = "no"?>
<!-- standalone = "no" indique que le document utilise des ressources exter-
nes -->
<!DOCTYPE exemple SYSTEM "DTDExemple.dtd">
<!-- Début du document -->
<exemple> Ceci est un exemple </exemple>
<!-- Fin du document -->
```

C'est la ligne **<!DOCTYPE exemple SYSTEM «DTDExemple.dtd»>** qui indique qu'on utilise une DTD externe.

Pour vérifier que ça fonctionne, affichez le document **exempleDTD.xml** dans votre navigateur.

Moi je vois ça :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!--  standalone = "no" indique que le document utilise des ressources exter-
nes  -->
  <!DOCTYPE exemple (View Source for full doctype...)>
```

```
<!--  Début du document   -->
  <exemple>Ceci est un exemple</exemple>
<!-- Fin du document -->
```

On peut lui rajouter une feuille de style.

Au-dessus ou en dessous de la ligne **<!DOCTYPE…**, rajoutez…

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

…et affichez à nouveau le document. Cette fois, on voit tout simplement ça :

```
Ceci est un exemple
```

5B. XML Schéma

Plus récente que les DTD, cette recommandation est aussi beaucoup plus riche.

Un exemple de schéma :

```
<?xml version = "1.0" encoding = " ISO-8859-1"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema">
    <xsd:element name = "exemple" type = "xsd:string"/>
</xsd:schema>
```

Voici un document XML respectant le schéma :

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="monStyle.css"?>
<exemple xmlns:xsi="http://www.w3.org/2000/10/XMLSchema"
    xsi:schemaLocation="schemaExemple.xsd"> Ceci est un exemple </exemple>
<!-- Fin du document -->
```

Je n'irai pas plus loin dans la présentation des schémas, mais au moins, si vous rencontrez sur le net un document xml auquel on a associé un schéma et pas une DTD, vous serez en mesure de vous en rendre compte.

5C. Comparaison DTD/schéma

• Une DTD n'est pas un document XML, un schéma est un document XML.

• Les schémas proposent de nombreux types de bases, contrairement aux DTD : le couple XML/XSD (XML Schema Definition) permet de définir un langage à typage fort de représentation des données, compatible avec les modèles relationnels et objet.

• Les DTD sont issues des technologies SGML et présentent quelques lacunes pour la validation des documents XML, largement utilisés actuellement dans les applications s'appuyant sur l'interopérablité (en utilisant des bases de données hétérogènes, par exemple, c'est-à-dire des bases de données créées avec différents SGBD), de communication réseau (flux XML) ou de publication web avancée.

Pour ma part, je vais vous présenter succinctement les DTD, mais pas les schémas xml, car il faudrait encore beaucoup trop de pages…

5D. Quelques syntaxes XML de base quand on utilise une DTD

Nous allons travailler sur le document xml **etudiants.xml,** afin de construire une DTD à laquelle ce document sera conforme.

5D1. L'élément racine

Si j'affiche le document **etudiants.xml** dans mon navigateur, sans lui associer de feuille de style, et que je clique sur tous les signes – qui précèdent toutes les balises composites, je vois ça :

```
<?xml version="1.0" encoding="ISO8859-1" standalone="yes" ?>
+ <btsIG>
+--+--+-- La balise btsIG est l'élément racine de mon document.
```

Donc, dans ma DTD, je dois indiquer cette racine.

<!ELEMENT btsIG…

5D2. Expression de la composition

Si, dans mon navigateur, je clique sur le + qui précède la balise **btsIG**, je vois ça :

```
<?xml version="1.0" encoding="ISO8859-1" standalone="yes" ?>
- <btsIG>
+ <etudiant>
--+ <etudiant>
--+ <etudiant>
--<!--  Et ainsi de suite pour chaque étudiant  -->
  </btsIG>
```

Notre balise racine **btsIG** est un élément composite. Elle peut contenir plusieurs balises **etudiant**.

Cela s'indique de cette manière dans la DTD :

```
<!ELEMENT btsIG (etudiant+)>
```

Ici, le signe + indique que l'élément **etudiant** doit figurer une ou plusieurs fois dans l'élément **btsIG**.

Si on veut que l'élément **etudiant** apparaisse zéro, une ou plusieurs fois, il faut écrire :

```
<!ELEMENT btsIG (etudiant*)>
```

Si on veut que l'élément etudiant apparaisse zéro ou une fois, il faut écrire :

```
<!ELEMENT btsIG (etudiant?)>
```

Et si on veut que l'élément **etudiant** apparaisse une et une seule fois, il faut écrire :

```
<!ELEMENT btsIG (etudiant)>
```

Dans le cas de notre exemple, les symboles adaptés sont + (si on veut qu'il y ait au moins un élément **etudiant**) et * (si on veut autoriser le fait qu'aucun élément **etudiant** ne soit présent dans l'élément **btsIG**).

L'élément **etudiant** est également un élément composite.

Si, dans mon navigateur, je clique sur le signe + placé, par exemple, devant la première balise **etudiant**, je vois ça :

```
- <etudiant>
  <nom>Tartempion</nom>
  <prenom>Gudule</prenom>
+ <option>
-  </etudiant>
```

Donc, ma DTD devient :

```
<!ELEMENT btsIG (etudiant+)>
<!ELEMENT etudiant (nom, prenom, option)>
```

Cette ligne rajoutée indique que l'élément **etudiant** est composé d'un seul élément **nom**, d'un seul élément **prenom**, et d'un seul élément **option**.

L'élément option est également un élément composite :

```
<nomoption>Réseau</nomoption>
+ <matieresprepondérantes>
  <profPrincipal>Madame Truc</profPrincipal>
```

Je complète donc ma DTD comme ça :

```
<!ELEMENT btsIG (etudiant+)>
<!ELEMENT etudiant (nom, prenom, option)>
<!ELEMENT option (nomoption, matieresprepondérantes, profprincipal)>
```

Et comme l'élément **matieresprepondérantes** est également composite :

```
<matiere1>Réseau</matiere1>
<matiere2>Système</matiere2>
```

Je complète encore ma DTD pour l'indiquer :

```
<!ELEMENT btsIG (etudiant+)>
<!ELEMENT etudiant (nom, prenom, option)>
<!ELEMENT option (nomoption, matieresprepondérantes, profprincipal)>
<!ELEMENT matieresprepondérantes (matiere1, matiere2)>
```

Bon, maintenant qu'on a indiqué la structure de tous les éléments composites, intéressons-nous aux éléments non composés, ceux qui contiennent les informations effectives, c'est à dire les données, qui sont, en fait, les feuilles terminales de l'arbre.

⚠ ***Lorsqu'on indique, dans un élément composite, les éléments qui le composent, il faut, dans le document xml qui se réfère à la DTD, que les éléments soient dans le même ordre que celui décrit dans la DTD, sinon, il n'est pas considéré comme valide.***

Remarque

Si on veut appliquer une même cardinalité à un groupe d'éléments, on doit utiliser les parenthèses, comme ça par exemple :

```
<!ELEMENT option (nomoption, matieresprepondérantes, profprincipal)*>
```

Ici, tout le groupe d'éléments peut apparaître de zéro à plusieurs fois.

```
<!ELEMENT option ((nomoption, matierespreponderantes)*, (profprincipal?))>
```

Ici, le premier groupe d'éléments se voit appliquer la cardinalité *, alors que l'élément **profprincipal** se voit appliquer la cardinalité **?**.

5D3. Les éléments contenant des données

Dans notre document **etudiants.xml**, les balises encadrant des données sont les balises **nom**, **prenom, nomoption, profprincipal, matiere1** et **matiere2**.

Pour que notre document xml soit valide par rapport à la DTD qu'on est en train de lui construire, pour chacun des éléments cités ci-dessus, on va écrire une ligne suivant la syntaxe suivante :

```
<!ELEMENT nomDeL'élément (#typeDeLaDonnée)>
```

Voilà ce que ça donne si je complète notre DTD :

```
<!ELEMENT btsIG (etudiant+)>
<!ELEMENT etudiant (nom, prenom, option)>
<!ELEMENT option (nomoption, matierespreponderantes, profprincipal)>
<!ELEMENT matierespreponderantes (matiere1, matiere2)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT nomoption (#PCDATA)>
<!ELEMENT profprincipal (#PCDATA)>
<!ELEMENT matiere1 (#PCDATA)>
<!ELEMENT matiere2 (#PCDATA)>
```

Ici, on n'utilise que le type prédéfini **PCDATA**, qui désigne une chaîne de caractères, mais il existe deux autres types de données prédéfinis :

- **ANY**, qui indique que l'élément peut contenir tout type de données ;
- **EMPTY,** qui indique que l'élément ne contient pas d'information (un peu comme l'élément **
** en html : cet élément ne contient pas d'information).

Nous allons maintenant voir que dans une DTD, on peut exprimer la présence alternative d'éléments dans un élément composite.

5D4. Ou exclusif entre des éléments

Reprenons notre exemple sur les étudiants. Supposons maintenant que notre DTD veuille n'autoriser la présence que d'une seule matière prépondérante sur les deux possibles.

Dans ce cas, notre DTD devient :

```
<!ELEMENT btsIG (etudiant+)>
<!ELEMENT etudiant (nom, prenom, option)>
<!ELEMENT option (nomoption, matierespreponderantes, profprincipal)>
<!ELEMENT matierespreponderantes (matiere1|matiere2)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT nomoption (#PCDATA)>
```

```
<!ELEMENT profprincipal (#PCDATA)>
<!ELEMENT matiere1 (#PCDATA)>
<!ELEMENT matiere2 (#PCDATA)>
```

Vous avez vu ? Pour exprimer la présence de l'un **ou** l'autre mais **pas les deux** éléments, on a remplacé la virgule séparant les deux éléments par une barre verticale.

Dans ce cas, notre document actuel **etudiants.xml** n'est pas valide par rapport à sa DTD, car nous avons 2 matières prépondérantes pour chaque étudiant.

Remarque

Si on avait voulu exprimer un ou inclusif entre les deux éléments (rappel : ou inclusif signifie l'un ou l'autre ou les deux), il aurait fallu utiliser les cardinalités exprimant que chaque élément doit être présent zéro ou une fois, comme ça :

```
<!ELEMENT matierespreponderantes (matiere1?, matiere2?)>
```

Remarque

Lorsqu'un document xml n'est pas valide, c'est-à-dire ne respecte pas la structure décrite dans sa DTD, le navigateur utilisé pour visualiser le document n'a aucune réaction. C'est normal… la DTD est plutôt utilisée par des programmes qui exploitent le contenu des documents xml.

5D5. Factorisation de certains éléments, description d'éléments avec attributs

Il est possible, dans les DTD, de créer des éléments réutilisables de factorisation, appelés entités. Une entité est utilisable autant de fois que l'on veut pour décrire une DTD.

La syntaxe, pour créer une entité, est la suivante :

```
<!ENTITY %nomDeL'entité description>
```

Ensuite, une entité se décrit exactement comme un autre élément (elle peut être composite, contenir des éléments de manière alternative etc.).

Dans la DTD, on peut ensuite indiquer cette entité (sans oublier le symbole %) à chaque fois que sa structure se présente.

Vous trouverez sur le net des explications simples et claires concernant les entités, notamment ici http://fr.selfhtml.org/xml/dtd/entites.htm.

La syntaxe des DTD permet également de décrire les éléments xml possédant des attributs. On peut même indiquer le type des attributs, s'ils sont obligatoires ou non… Allez voir ici : http://fr.selfhtml.org/xml/dtd/attributs.htm.

Moi, je vais m'arrêter là en ce qui concerne les DTD, car ce cours de POW va finir par être un trop gros pavé. Eh ben oui, voilà, 25 pages sur xml alors que je m'étais promis de ne pas dépasser les 15 pages… Mais face à ces technologies si touffues, il est difficile de faire très court, alors là, maintenant, stop, j'arrête, c'est fini… Il ne me reste plus qu'à conclure.

6. En conclusion, des pistes pour aller un peu plus loin

Pour prolonger cette initiation à xml et à toutes les technologies qui lui sont liées, ceux qui sont intéressés peuvent compléter leur connaissance sur les DTD.

Et également s'initier :

– aux feuilles de style **xsl** ;

– aux **schémas** xml ;

– à l'utilisation de liens hypertexte dans les documents xml avec **xlink** ;

– aux requêtes sur des documents xml à l'aide du langage **xquery** ;

– aux fonctions de **php** permettant la manipulation de documents **xml** ;

– au langage **xhtml**…

Les technologies et outils autour de xml ne manquent pas.

Me voilà prête pour la conclusion générale…

Conclusion générale

Pour carrément prolonger ce cours d'initiation à la programmation web, vous pouvez :

– parfaire vos connaissances de javascript, html, php, xml ;

– vous mettre à ASP de microsoft ;

– vous initier à la technologie flash de macromedia ;

– apprendre à écrire des applets java ;

– vous intéresser à l'aspect sécuritaire de la programmation web (protocoles sécurisés : https, ssl), cryptage, techniques de clés (privées, publiques) etc. etc.

N'oubliez pas également de vous tenir au courant des droits et devoirs en matière de publication de sites sur le net. Pour cet aspect juridique de la conception de sites, je vous renvoie au site de la CNIL et à tout cours de droit de l'informatique.

Bonne continuation et à bientôt.