



Structurez vos données avec XML

Par Ludovic ROLAND (Wapiti89)



*Licence Creative Commons 4 2.0
Dernière mise à jour le 20/12/2012*

Sommaire

Sommaire	2
Partager	2
Structurez vos données avec XML	4
Partie 1 : Les bases du XML	4
Qu'est-ce que le XML ?	5
Qu'est ce que le XML ?	5
Première définition	5
Une nouvelle définition	6
Origine et objectif du XML	6
La petite histoire du XML	6
Les objectifs du XML	6
En résumé	6
Les bons outils	6
L'éditeur de texte	7
Sous Windows	7
Sous GNU/Linux	8
Sous MAC OS X	9
EditiX	10
La version payante	10
La version gratuite	10
La mise en place sous GNU/Linux	11
<oxygen/> XML Editor	13
Les éléments de base	14
Les balises	15
Les balises par paires	15
Les balises uniques	16
Les règles de nommage des balises	16
Les attributs	17
Définition	17
Quelques règles	17
Les commentaires	17
En résumé	18
Votre premier document XML	19
Structure d'un document XML	19
Le prologue	19
Le corps	19
Un document complet	20
Un document bien formé	20
Utilisation d'EditiX	21
Créer un nouveau document	21
Vérification du document	22
L'indentation	23
L'arborescence du document	23
Enregistrer votre document	23
TP : structuration d'un répertoire	23
L'énoncé	24
Une solution	24
Quelques explications	25
Partie 2 : Créez des définitions pour vos documents XML	25
Introduction aux définitions et aux DTD	26
Les éléments	26
La syntaxe	26
Retour sur la balise	26
Retour sur le contenu	26
Structurer le contenu des balises	29
La séquence	29
La liste de choix	30
La balise optionnelle	31
La balise répétée optionnelle	31
La balise répétée	32
En résumé	33
DTD : les attributs et les entités	33
Les attributs	34
La syntaxe	34
Retour sur la balise et l'attribut	34
Retour sur le type	34
Retour sur le mode	37
Les entités	39
Définition	39
Les entités générales	39
Les entités paramètres	40
Les entités externes	41
En résumé	42
DTD : où les écrire ?	42

Les DTD internes	43
Définition	43
La Syntaxe	43
Illustrons avec un exemple	43
Les DTD externes	44
Définition	44
La Syntaxe	45
Retour sur le prologue	47
Un exemple avec EditiX	47
Création du document XML	47
Création du document DTD	48
Vérification de la DTD	49
Vérification du document XML	50
En résumé	50
TP : définition DTD d'un répertoire	51
L'énoncé	51
Une solution	52
Un bref commentaire	53



Structurez vos données avec XML

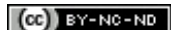
Par



Ludovic ROLAND (Wapiti89)

Mise à jour : 20/12/2012

Difficulté : Facile



13 061 visites depuis 7 jours, classé 22/798

Vous souhaitez structurer les données manipulées ou échangées par vos programmes ou vos applications mobiles ?

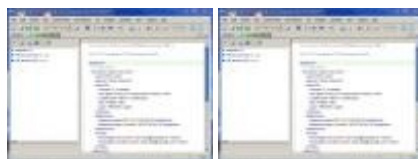
Ne cherchez plus ! XML va vous faciliter la vie !

XML est un langage de balisage générique qui permet de structurer des données afin qu'elles soient lisibles aussi bien par les humains que par des programmes de toute sorte. Il est souvent utilisé pour faire des échanges de données entre un programme et un serveur ou entre plusieurs programmes.

Pour vous donner un exemple concret, prenons un exemple d'actualité : celui d'une application téléphonique qui met à jour les données qu'elle contient. L'application demande à un serveur web les dernières informations dont il dispose. Après être allé les chercher, ce dernier doit les communiquer. C'est là qu'intervient le XML. Le serveur web se sert du XML pour structurer les informations qu'il doit renvoyer à l'application téléphonique. Lorsque cette dernière reçoit les informations ainsi structurées, elle sait comment les lire et les exploiter rapidement !

Dans ce cours destiné aux **débutants**, nous découvrirons ensemble comment écrire des documents XML. Puisque cette partie ne nous prendra pas énormément de temps, nous en profiterons pour découvrir ensemble tout ce qui tourne autour de l'univers du XML. Ainsi, nous verrons également comment :

- imposer une structure bien précise à nos documents XML ;
- les mettre en forme ;
- lire facilement les données contenues dans un document XML ;
- transformer les documents XML vers d'autres formats comme une page internet ou un fichier PDF ;



Aperçu de fichiers que nous aurons l'occasion d'écrire dans ce tutoriel

Vous l'aurez compris, le programme s'annonce chargé. C'est pourquoi je vous propose de commencer tout de suite ! 😊

Partie 1 : Les bases du XML

Dans cette première partie, nous allons commencer par le tout début à savoir la structure d'un document XML.

Mais avant ça, nous allons nous attarder sur l'origine du XML, à quoi il sert et sur les bons outils à installer pour appréhender correctement ce tutoriel.

Qu'est-ce que le XML ?

Voici donc le tout premier chapitre de la longue série que va comporter ce tutoriel sur XML !

Tout au long de ce cours nous parlerons du XML et des technologies qui gravitent autour. Bien que dans l'introduction, je vous ai déjà donné un exemple d'utilisation du XML, je vous propose de revenir un peu plus en détail sur cette technologie, son origine et son objectif.

Qu'est ce que le XML ? Première définition

Citation

Le XML ou eXtensible Markup Language est un langage informatique de balisage générique.

Cette définition est à mes yeux un peu barbare et technique. C'est pourquoi je vous propose de décortiquer les mots clefs.

Un langage informatique

Je suis sûr que vous n'êtes pas sans savoir qu'en informatique, il existe plusieurs centaines de langages destinés à des utilisations très diverses.

En vulgarisant un peu (beaucoup), il est possible de les regrouper dans 3 grosses catégories :

- les langages de programmation ;
- les langages de requête ;
- les langages de description;

Les **langages de programmation** permettent de créer des programmes, des applications mobiles, des sites internet, des systèmes d'exploitation, etc. Certains langages de programmation sont extrêmement populaires. En Mai 2012, les 10 langages de programmation les plus populaires sont le C, le Java, le C++, l'Objective-C, le C#, le PHP, le Basic, le Python, le Perl et le Javascript.

Les **langages de requêtes** permettent quant à eux d'interroger des structures qui contiennent des données. Parmi les langages de requête les plus connus, on peut par exemple citer le SQL pour les bases de données relationnelles, le SPARQL pour les graphes RDF et les ontologies OWL ou encore le XQuery pour les documents XML.

Finalement, les **langages de description** permettent de décrire et structurer un ensemble de données selon un jeu de règles et des contraintes définies. On peut par exemple utiliser ce type de langage pour décrire l'ensemble des livres d'une bibliothèque, ou encore la liste des chansons d'un CD, etc. Parmi les langages de description les plus connus, on peut citer le SGML, le XML ou encore le HTML.

Un langage de balisage générique

Un langage de balisage est un langage qui s'écrit grâce à des balises. Ces balises permettent de structurer de manière hiérarchisée et organisée les données d'un document.

Si vous ne savez pas ce qu'est une balise, ne vous inquiétez pas, nous reviendrons sur ce terme un peu plus loin dans le tutoriel.



Finalement, le terme **générique** signifie que nous allons pouvoir créer nos propres balises. Nous ne sommes pas obligé d'utiliser un ensemble de balises existantes comme c'est par exemple le cas en HTML.

Une nouvelle définition

Suite aux explications du dessus, je vous propose que l'on écrive une nouvelle définition du langage XML bien moins technique que la première donnée dans ce cours.

Voici celle que je vous propose : **le langage XML est un langage qui permet de décrire des données à l'aide de balises et de règles que l'on peut personnaliser.**

Je me doute que certaines notions peuvent vous sembler abstraites, mais ne vous inquiétez pas, tout sera expliqué dans la suite de ce tutoriel. 😊

Origine et objectif du XML

La petite histoire du XML

Vous vous doutez bien que le langage XML n'a pas été créé pour s'amuser. En effet, sa création avait pour objectif de répondre à un besoin très précis : **l'échange de données.**

Dans les débuts d'internet, les ordinateurs et les programmes échangeaient des données via des fichiers. Mais malheureusement, ces fichiers avaient bien souvent des règles de formatage qui leur étaient propres. Par exemple, les données étaient séparées des points, des virgules, des espaces, des tirets, etc.

Le problème avec ce système est qu'il fallait sans cesse adapter les programmes au format du fichier ce qui représentait une charge de travail importante.

Il a donc fallu régler ce problème assez vite. Le langage **SGML** ou **Standard Generalized Markup Language** est alors né. C'est un langage puissant, extensible et standard qui permet de décrire à l'aide de balises un ensemble de données. Le problème est que ce langage est très complexe n'est pas forcément compatible pour effectuer des échanges sur le web.

Un groupe d'informaticiens ayant de l'expérience dans le SGML et le web a alors décidé de se pencher sur le sujet. Le langage **XML** est né. Le XML 1.0 est devenu une recommandation du W3C le 10 février 1998.

Depuis, les spécifications du langage ont évolué, et la version 1.1 est publiée le 4 février 2004. C'est pourtant la version 1.0 du langage XML qui est la plus utilisée encore aujourd'hui et c'est cette version que nous allons étudier au cours de ce tutoriel.

Les objectifs du XML

Comme nous l'avons vu, l'objectif du XML est de pouvoir faciliter les échanges de données entre les machines. Mais à ça s'ajoute un autre objectif important : **décrire les données de manière aussi bien compréhensible par les Hommes qui écrivent les documents XML que par les machines qui les exploitent.**

Le XML se veut également compatible avec le web afin que les échanges de données puissent se faire facilement à travers le réseau internet.

Finalement, le XML se veut standardisé, simple, mais surtout extensible et configurable afin que n'importe quel type de données puisse être décrit.

En résumé

Pour résumer ce premier chapitre, peu technique mais très important, nous retiendrons que le XML est un langage de balisage générique qui permet de structurer des données dans l'objectif de les partager.

Les bons outils

Maintenant que vous êtes un peu plus familier avec le XML et que son utilité est plus claire dans vos esprits, je vous propose de jeter un coup d'œil à certains outils qui pourront nous être utiles pour rédiger nos documents XML.

Les outils proposés dans ce tutoriel, nous permettront d'être plus productif. En effet, tous proposent des fonctions clés pour gagner en productivité comme par exemple une fonction de coloration syntaxique, de validation, de vérification ou encore d'exploitation.

Je ne peux que vous encourager à les essayer et d'adopter celui qui vous correspond le plus !

L'éditeur de texte

Il faut savoir que dans le fond, un document XML n'est en réalité qu'un simple document texte.

C'est pourquoi, il est tout à fait possible d'utiliser un éditeur de texte pour la rédaction de nos documents XML !

Sous Windows

Sous Windows, un éditeur de texte portant le nom de **Bloc-notes** est généralement installé par défaut.

En théorie il est suffisant et fonctionnel. Dans la pratique, les fonctionnalités qu'il offre sont limitées et des options comme la coloration syntaxique ou la numérotation des lignes manquent.

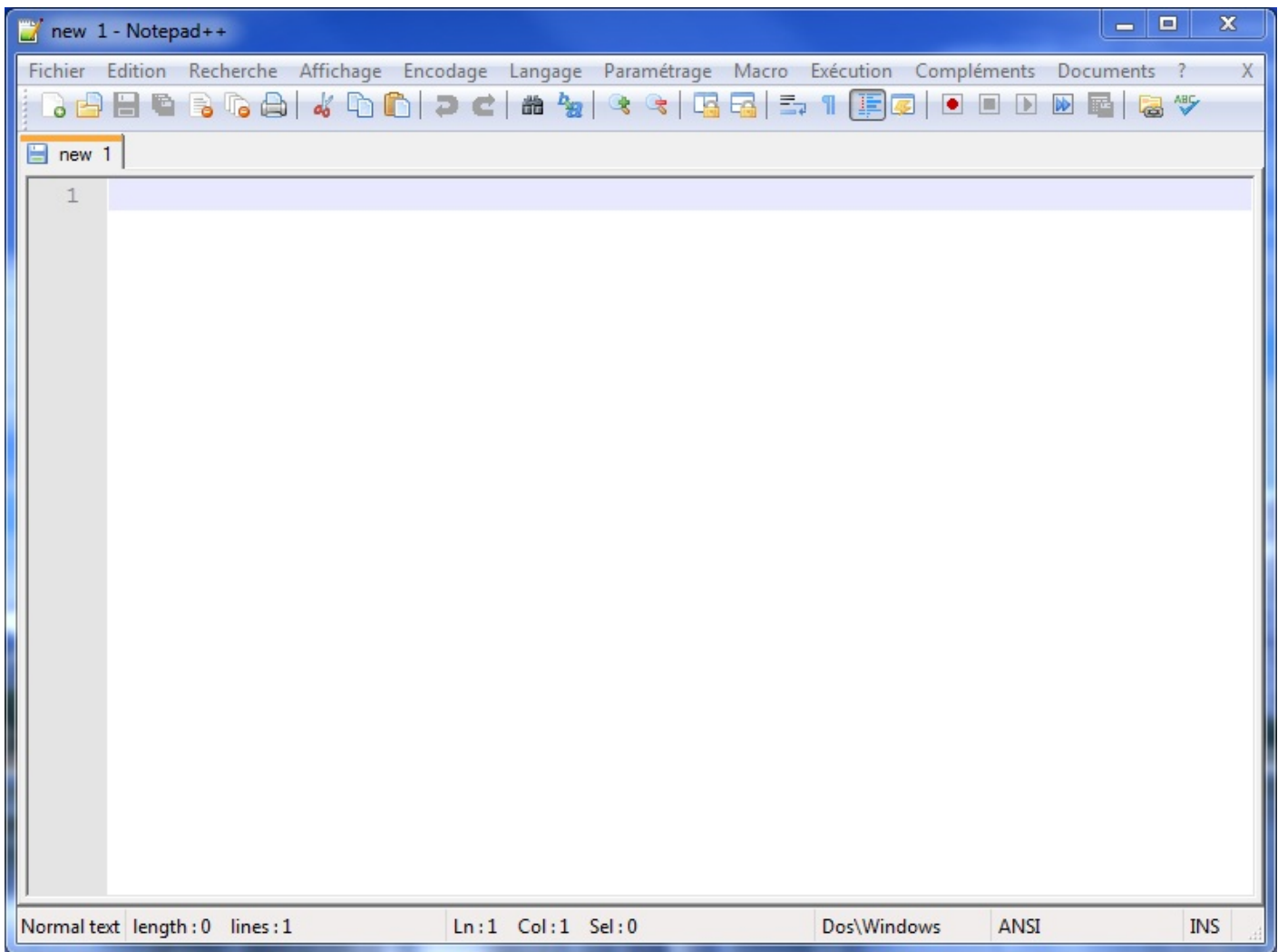
Je vous propose donc d'utiliser **Notepad++** qui est parfait pour ce que nous voulons faire puisqu'il permet de pallier les manques du Bloc-notes.

Il s'agit d'un logiciel gratuit, n'hésitez donc pas une seule seconde à le télécharger ! 😊

Télécharger Notepad++

Je ne vais pas détailler la procédure d'installation qui est classique pour un logiciel tournant sous Windows.

Une fois installé, lancez le logiciel. Vous devriez avoir obtenu la fenêtre suivante :



Afin d'adapter la coloration syntaxique au langage XML, ils vous suffit de sélectionner **Langage** dans la barre de menu puis **XML** dans la liste.

Lorsque vous enregistrerez vos documents, il suffira alors de préciser comme extension **".xml"** pour conserver la coloration syntaxique d'une fois sur l'autre.

Sous GNU/Linux

Par défaut, les distributions Linux sont souvent livrées avec de très bons éditeurs de texte.

Si vous aimez la console, vous pouvez par exemple utiliser **nano**, **emacs**, **vi** ou encore **vim**.

Si vous préférez les interfaces graphiques, je vous conseille d'utiliser l'excellent **gedit** qui normalement doit-être installé par défaut.

Si jamais ce n'est pas le cas, la commande suivante vous permettra de l'installer en quelques instants :

Code : Console

```
sudo apt-get install gedit
```

Une fois ouvert vous devriez avoir quelque chose comme ça :

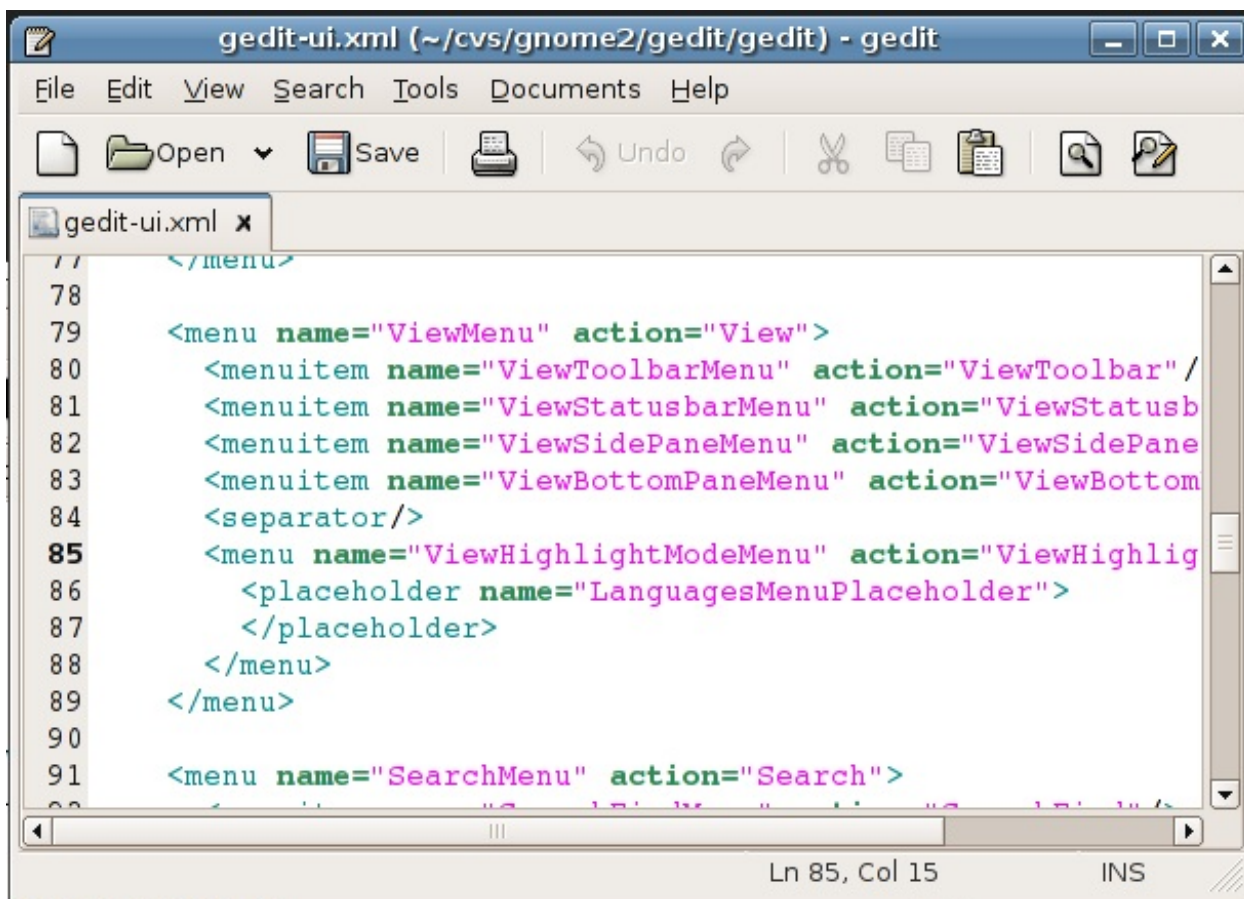


Photo issue du site officiel du projet gedit

Afin d'adapter la coloration syntaxique au langage XML, ils vous suffit de sélectionner **Affichage** dans la barre de menu puis **Mode de coloration** et finalement choisir le **XML** dans la liste.

Lorsque vous enregistrerez vos documents, il suffira alors de préciser comme extension **".xml"** pour conserver la coloration syntaxique d'une fois sur l'autre.

Sous MAC OS X

Pour les utilisateurs du système d'exploitation d'Apple, je vous conseille de vous tourner vers **jEdit**.

Vous pouvez le télécharger [ici](#).

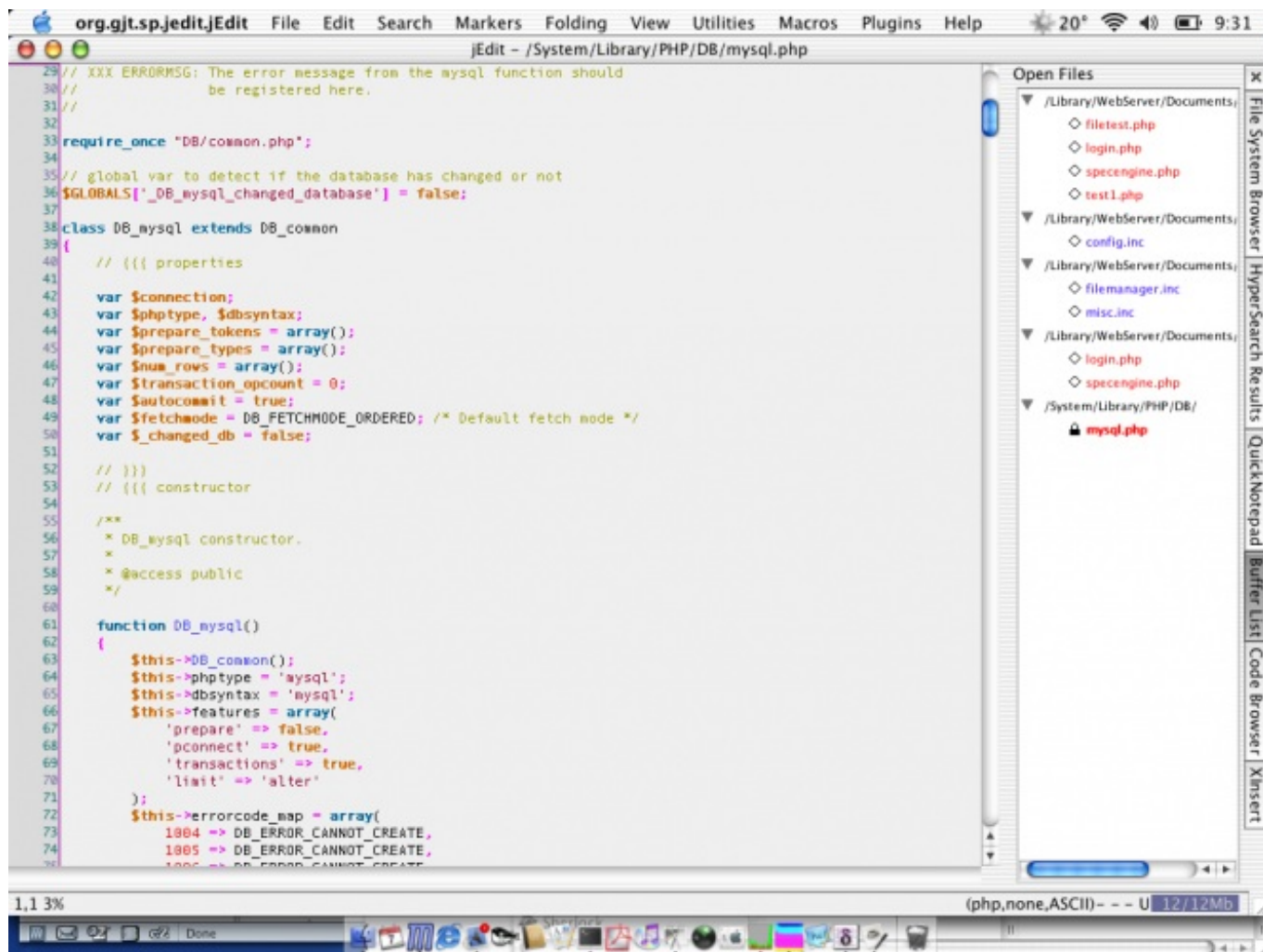


Photo issue du site officiel du projet jEdit

Editix

Editix est un éditeur XML qui fonctionne sur les plateformes Windows, GNU/Linux ou Mac OS X.

En plus de la coloration syntaxique essentielle à l'écriture de documents XML, ce logiciel nous offre tout un tas d'outils qui nous seront utiles dans la suite de ce tutoriel comme par exemple la validation des documents.

La version payante

Il existe plusieurs versions de ce logiciel. La dernière en date est **Editix 2012**.

Cette version complète est payante, mais plusieurs licences sont disponibles. Les étudiants peuvent par exemple bénéficier de 50% de réduction.

Télécharger Editix 2012

La version gratuite

Heureusement pour les pauvres Zéros fauchés que nous sommes, une version gratuite existe !

Il s'agit d'une version allégée d'**Editix 2010**. Notez bien que cette version est réservée à un usage non-commercial !

Télécharger Editix 2010, Lite Version

Puisque c'est cette version que je vais en partie utiliser dans la suite du tutoriel, je vous propose de détailler la procédure de mise

en place du logiciel sous GNU/Linux.

Je ne détaille pas la procédure d'installation sous Windows et MAC OS X puisqu'elle est des plus classique.

La mise en place sous GNU/Linux

Téléchargement et installation

Enregistrez l'archive sur votre bureau puis lancez votre plus beau terminal afin de débiter la procédure.

Commencez par déplacer l'archive **editix-free-2010.tar.gz** fraîchement téléchargée dans le répertoire **/opt/** via la commande :

Code : Console

```
sudo mv ~/Bureau/editix-free-2010.tar.gz /opt/
```

Déplacez vous maintenant dans le dossier **/opt/** via la commande :

Code : Console

```
cd /opt/
```

Nous allons maintenant extraire les fichiers de l'archive que nous avons téléchargée. Pour se faire, vous pouvez utiliser la commande :

Code : Console

```
sudo tar xvzf editix-free-2010.tar.gz
```

Un dossier nommé **editix** doit alors être apparu. Il contient les fichiers que nous venons d'extraire.

Vous pouvez alors supprimer l'archive via la commande :

Code : Console

```
sudo rm editix-free-2010.tar.gz
```

On pourrait choisir de s'arrêter là et de lancer le logiciel en ligne de commande se rendant dans le répertoire **/opt/editix/bin/** pour en exécutant le script **run.sh** via la commande :

Code : Console

```
./run.sh
```

Mais pour plus de confort, je vous propose de créer un **launcher**.

Création du launcher

Pour se faire, faites un clic droit sur le menu **Applications** de votre tableau bord (ou sur le **Menu** si vous êtes sur une LMDE par exemple) et cliquez sur **Editer le menu** .

Dans la colonne de gauche choisissez le menu **Programmation** puis cliquez sur le bouton **Nouvel élément** dans la colonne de droite.

La fenêtre suivante devrait alors s'afficher :



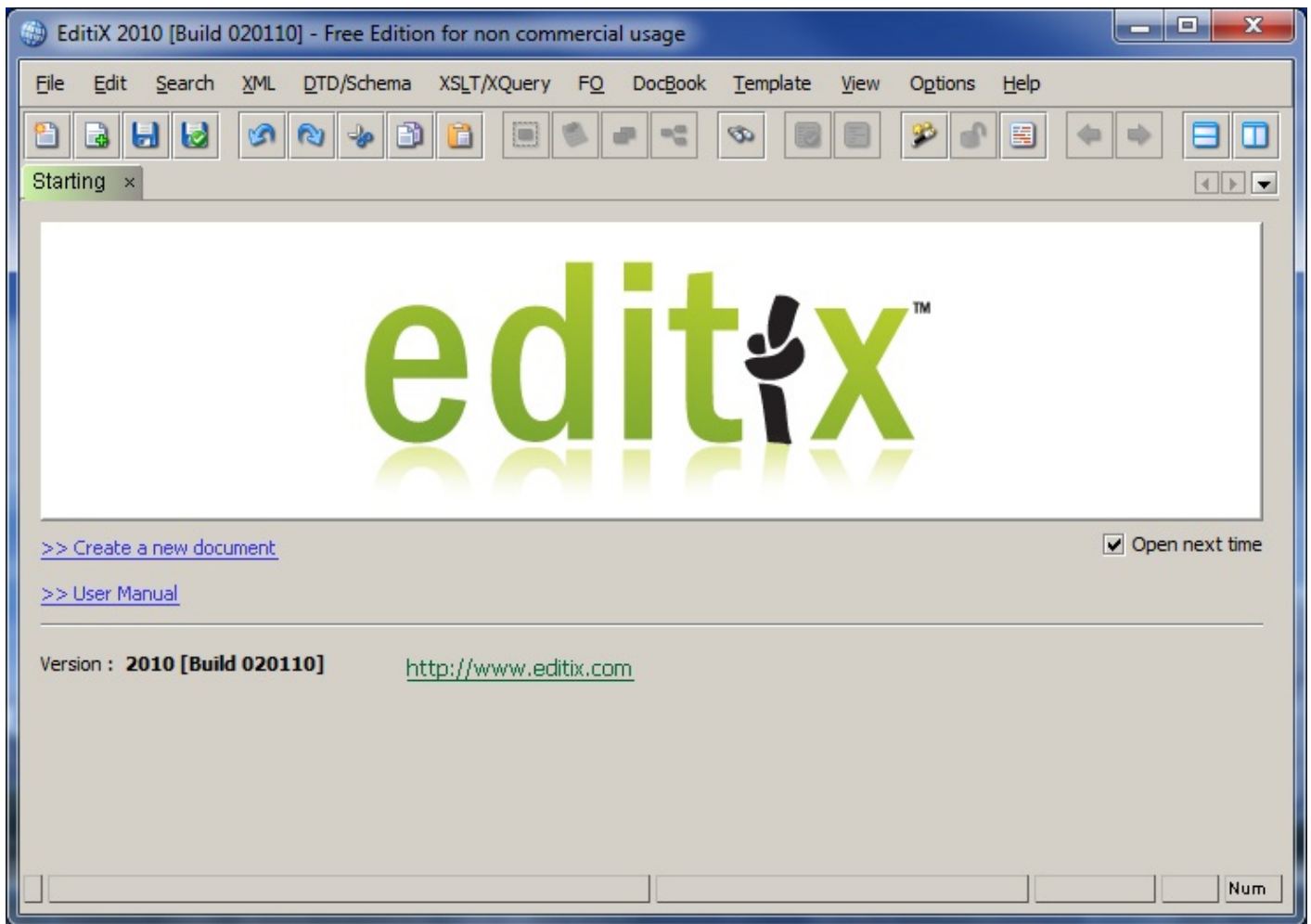
Remplissez le formulaire avec les informations suivantes :

- Type : Application
- Nom : Editix
- Commande : /opt/editix/bin/run.sh

Finalement, finalisez la création du launcher en cliquant sur le bouton **Valider**.

Editix devrait maintenant apparaître dans vos applications et plus particulièrement dans le menu programmation.

Quelque soit votre système d'exploitation, voici à quoi doit ressembler la fenêtre du logiciel après son lancement :



<oXygen/> XML Editor

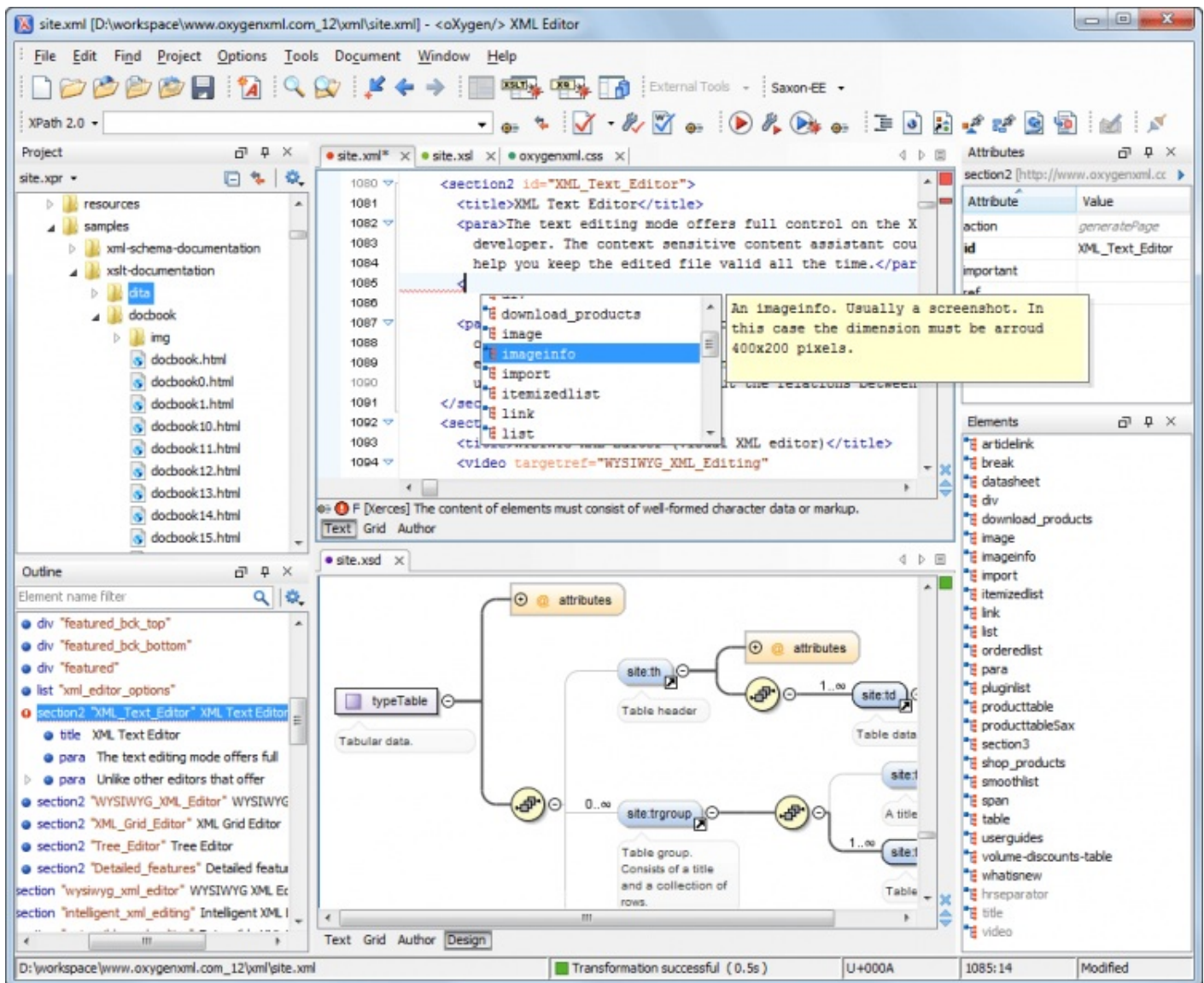
Je vais conclure cette présentation avec le logiciel <oXygen/> XML Editor qui comme Editix est multiplateformes.

Il n'existe pas de version gratuite du logiciel, mais il reste possible de le tester gratuitement pendant 30 jours.

Comme pour Editix, <oXygen/> XML Editor propose plusieurs types de licences. Ainsi, les étudiants peuvent obtenir des réductions très intéressantes.

Télécharger <oXygen/> XML Editor

Voici un exemple d'écran récupéré sur le site internet de l'éditeur du logiciel.



Maintenant que vous êtes équipé jusqu'aux dents, nous sommes prêts à commencer notre étude du langage XML ! 🦷

Les éléments de base

C'est maintenant que la pratique commence ! Dans ce chapitre, nous allons découvrir ensemble les bases du XML.

Au programme donc de ce chapitre :

- les balises ;
- les attributs ;
- les commentaires.

Les balises

Dans le tout premier chapitre, je vous définissais le langage XML comme un langage informatique de balisage.

En effet, les balises sont les éléments de base d'un document XML. Une balise porte un nom qui est entouré de **chevrons**. Une balise commence donc par un < et se termine par un >. Par exemple : `<balise>` définit une balise qui s'appelle "**balise**".

En XML, on distingue 2 types de balises : les **balises par paires** et les **balises uniques**.

Les balises par paires

Définition

Les balises par paires sont composées en réalité de 2 balises que l'on appelle **ouvrantes** et **fermantes**.

La balise ouvrante commence par < et se termine par > tandis que la balise fermante commence par </ et se termine par >.

Par exemple :

Code : XML

```
<balise></balise>
```



Il est extrêmement important que les balises ouvrantes et fermantes aient **exactement** le même nom. XML est sensible à la casse (c'est-à-dire qu'il fait la distinction entre les majuscules et les minuscules) !

Toute balise ouverte doit impérativement être fermée. C'est une règle d'or !

Bien évidemment, on peut mettre "des choses" entre ces balises. On parle alors de **contenu**.

Par exemple :

Code : XML

```
<balise>Je suis le contenu de la balise</balise>
```

Quelques règles

Une balise par paires ne peut pas contenir n'importe quoi.

Elle peut contenir une **valeur simple** comme par exemple une chaîne de caractères, un nombre entier, un nombre décimal, etc.

Code : XML

```
<balise1>Ceci est une chaîne de caractères</balise1>
<balise2>10</balise2>
<balise3>7.5</balise3>
```

Une balise en paires peut également contenir une **autre balise**. On parle alors d'**arborescence**.

Code : XML

```
<balise1>
  <balise2>10</balise2>
</balise1>
```

Faites cependant très attention, si une balise peut en contenir une autre, il est cependant interdit de les chevaucher. L'exemple suivant n'est pas du XML !



Code : XML

```
<balise1><balise2>Ceci est une chaîne de
caractères</balise1></balise2>
```

Finalement, une balise en paires peut contenir un mélange de valeurs simples et de balises comme en témoigne l'exemple suivant :

Code : XML

```
<balise1>
  Ceci est une chaîne de caractères
  <balise2>10</balise2>
  7.5
</balise1>
```

Les balises uniques

Une balise unique est en réalité une balise en paires qui n'a pas de contenu.

Vous le savez, les informaticiens sont des fainéants. Ainsi, plutôt que de perdre du temps à ouvrir et fermer des balises sans rien écrire entre, une syntaxe un peu spéciale a été mise au point :

Code : XML

```
<balise />
```

Les règles de nommage des balises

Ce qui rend le XML générique, c'est la possibilité de créer votre propre langage balisé. Ce langage balisé, comme son nom l'indique, est un langage composé de balises sauf qu'en XML, c'est vous qui choisissez leurs noms.

L'exemple le plus connu des langages balisés de type XML est très certainement le xHTML qui est utilisé dans la création de sites internet.

Il y a cependant quelques règles de nommage à respecter pour les balises de votre langage balisé :

- les noms peuvent contenir des lettres, des chiffres ou des caractères spéciaux ;
- les noms ne peuvent pas débiter par un nombre ou un caractère de ponctuation ;
- les noms ne peuvent pas commencer par les lettres XML (quelque soit la casse) ;
- les noms ne peuvent pas contenir d'espaces ;
- on évitera les caractères -, ;, . < et > qui peuvent être mal interprétés dans vos programmes.

Les attributs

Définition

Il est possible d'ajouter à nos balises ce qu'on appelle des **attributs**. Tout comme pour le nom des balises, c'est vous qui en choisissez le nom.

Un attribut peut se décrire comme une option ou une donnée cachée. Ce n'est pas l'information principale que souhaite transmettre la balise, mais il donne des renseignements supplémentaires sur son contenu.

Pour que ce soit un peu plus parlant, voici tout de suite un exemple :

Code : XML

```
<prix devise="euro">25.3</prix>
```

Dans l'exemple ci-dessus, l'information principale est le prix. L'attribut `devise` nous permet d'apporter des informations supplémentaires sur ce prix, mais ce n'est pas l'information principale que souhaite transmettre la balise `<prix/>`.

Une balise peut contenir 0 ou plusieurs attributs. Par exemple :

Code : XML

```
<prix devise="euro" moyen_paiement="chèque">25.3</prix>
```

Quelques règles

Tout comme pour les balises, quelques règles sont à respecter pour les attributs :

- les règles de nommage sont les mêmes que pour les balises ;
- la valeur d'un attribut doit impérativement être délimitée par des guillemets simples ou doubles ;
- dans une balise, un attribut ne peut-être présent qu'une seule fois.

Les commentaires

Avant de passer à la création de notre premier document XML, j'aimerais vous parler des **commentaires**.

Un commentaire est un texte qui permet de donner une indication sur ce que l'on fait. Ils vous permettent d'annoter votre fichier et d'expliquer une partie de celui-ci.

En XML, les commentaires ont une syntaxe particulière. C'est une balise unique qui commence par `<!--` et qui se termine par `-->`.

Code : XML

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

Voyons tout de suite sur un exemple concret :

Code : XML

```
<!-- Description du prix -->  
<prix devise="euro">12.5</prix>
```

C'est sûr que sur cet exemple les commentaires semblent un peu inutiles... Mais je vous assure qu'ils vous seront d'une grande aide pendant la rédaction de longs documents XML ! 😊

En résumé

Pour résumer ce chapitre, nous venons de voir que 2 types de balises existent : les balises par paires, qui s'ouvrent et se ferment et les balises uniques que l'on peut considérer comme une balise par paires sans contenu.

Au cours de ce chapitre, nous avons également vu que les balises peuvent contenir des attributs et que de manière générale, un document XML peut contenir des commentaires.

Votre premier document XML

Jusqu'à maintenant, nous avons découvert les éléments de base du XML, mais vous ignorez encore comment écrire un document XML. Ne vous inquiétez pas, ce chapitre a pour objectif de corriger ce manque !

Dans ce chapitre, je vous propose donc de s'attaquer tout ce qui se rattache à l'écriture d'un document XML. Nous reviendrons sur la structure générale d'un document et de nouvelles notions clefs avant d'utiliser le logiciel EditiX pour écrire notre premier document XML en bonne et due forme !

Structure d'un document XML

Un document XML peut être découpé en 2 parties : le **prologue** et le **corps**.

Le prologue

Il s'agit de la première ligne de votre document XML. Il donne des informations de traitement.

Voici à quoi va ressembler notre prologue dans cette première partie du tutoriel :

Code : XML

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

Comme vous pouvez le remarquer, le prologue est une balise unique qui commence par **<?xml** et qui se termine par **?>**.

La version

Dans le prologue, on commence généralement par indiquer la version de XML que l'on utilise pour décrire nos données. Pour rappel, il existe actuellement 2 versions : 1.0 et 1.1.

A noter que le prologue n'est obligatoire que depuis la version 1.1, mais il est plus que conseillé de le mettre quand même lorsque vous utilisez la version 1.0.

La différence entre les 2 versions est une amélioration dans le support des différentes versions de l'Unicode. Sauf si vous souhaitez utiliser des caractères chinois dans vos documents XML, il conviendra d'utiliser la version 1.0 qui est encore aujourd'hui la version la plus utilisée.

Le jeu de caractères

La seconde information de mon prologue est `encoding="UTF-8"`.

Il s'agit du jeu de caractères utilisé dans mon document XML. Par défaut, l'encodage de XML est l'UTF-8, mais si votre éditeur de texte enregistre vos documents en ISO8859-1, il suffit de la changer dans le prologue :

Code : XML

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

Un document autonome

La dernière information présente dans le prologue est `standalone="yes"`.

Cette information permet de savoir si votre document XML est autonome ou si un autre document lui est rattaché.

Il est encore un peu tôt pour vous en dire plus. Nous reviendrons sur cette notion dans la partie 2 du tutoriel. Pour le moment acceptez le fait que nos documents sont tous autonomes.

Le corps

Le corps d'un document XML est constitué de l'ensemble des balises qui décrivent les données.

Il y a cependant une règle très importante à respecter dans la constitution du corps : **une balise en paires unique doit contenir toutes les autres**.

Cette balise est appelée **élément racine** du corps.

Voyons tout de suite un exemple :

Code : XML

```
<racine>
  <balise_paire>texte</balise_paire>
  <balise_paire2>texte</balise_paire2>
  <balise_paire>texte</balise_paire>
</racine>
```

Bien évidemment, lorsque vous créez vos documents XML, le but est d'être le plus explicite possible dans le nommage de vos balises. Ainsi, le plus souvent, la balise racine aura pour mission de décrire ce qu'elle contient.

Si je choisis de décrire un répertoire, je peux par exemple nommer mes balises comme dans l'exemple suivant :

Code : XML

```
<repertoire>
  <personne>Bernard</personne>
  <personne>Patrick</personne>
</repertoire>
```

Un document complet

Un document XML certes simple mais complet pourrait donc être le suivant :

Code : XML

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes" ?>
<repertoire>
  <personne>Bernard</personne>
  <personne>Patrick</personne>
</repertoire>
```

Un document bien formé

Quand vous entendrez parler de XML, vous entendrez souvent parler de **document XML bien formé** ou **well-formed** en anglais.

Cette notion décrit en fait un document XML conforme aux règles syntaxiques décrites tout au long de cette première partie du tutoriel.

On peut résumer un document XML bien formé à un document XML avec une syntaxe correcte, c'est-à-dire :

- s'il s'agit d'un document utilisant la version 1.1 du XML, le prologue est bien renseigné ;
- le document XML ne possède qu'une seule balise racine ;
- le nom des balises et des attributs est conforme aux règles de nommage ;
- toutes les balises en paires sont correctement fermées ;
- toutes les valeurs des attributs sont entre guillemets simples ou doubles ;

- les balises de votre document XML ne se chevauchent pas, il existe une arborescence dans votre document.

Si votre document XML est bien formé, félicitation, il est exploitable ! Dans le cas contraire, votre document est inutilisable.

Utilisation d'EditiX

Alors c'est bien beau, mais depuis le début on parle du XML, mais finalement nous n'avons toujours pas utilisé les logiciels du chapitre 2.

Dans cette partie, nous allons créer notre premier document XML grâce au logiciel EditiX et vérifier qu'il est bien formé.

Pour cet exemple, je vous propose d'utiliser le document complet que nous avons construit juste au dessus :

Code : XML

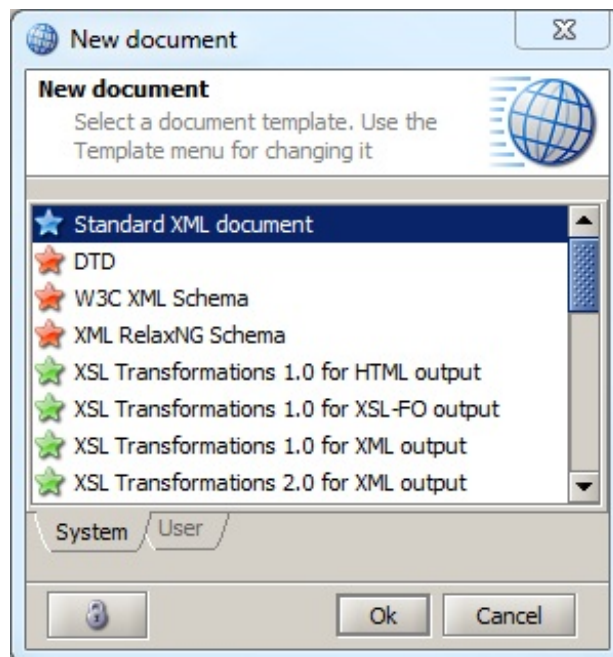
```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<repertoire>
  <personne>Robert</personne>
  <personne>John</personne>
</repertoire>
```

Créer un nouveau document

Commencez par lancer EditiX.

Pour créer un nouveau document, vous pouvez cliquer sur l'icône , sélectionner dans la barre de menu **File** puis **New** ou encore utiliser le raccourci clavier Ctrl + N.

Dans la liste qui s'affiche, sélectionnez **Standard XML document**.



Surprise ! Votre document XML n'est pas vierge. Voici ce que vous devriez avoir :

Code : XML

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

```
<!-- New document created with EditiX at Fri May 18 00:11:02 CEST  
2012 -->
```


Comme vous pouvez le constater, EditiX s'est chargé pour vous d'écrire le prologue de notre document XML. Il s'est également chargé d'écrire un petit commentaire pour vous rappeler la date et l'heure de création de votre document.

Puisque notre document sera autonome, vous pouvez modifier le prologue pour l'indiquer :

Code : XML

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

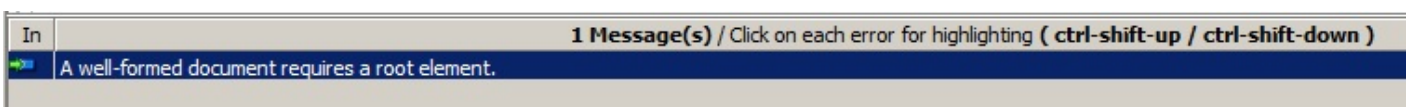
Vérification du document

Nous pouvons vérifier dès maintenant si notre document est bien formé. Pour se faire, vous pouvez cliquer sur l'icône , sélectionner dans la barre de menu **XML** puis **Check this document** ou encore utiliser le raccourci clavier Ctrl + K.

Vous devriez alors avoir une erreur. La ligne où se situe l'erreur est représentée par un rectangle aux bords rouges sur notre espace de travail :



Nous avons donc une erreur à la ligne 6 de notre document. Pour en savoir plus sur notre erreur, il suffit de regarder en bas de l'écran, voici ce que vous devriez avoir :



Pour ceux qui ne parle pas anglais, voici ce que dit le message :

Citation : EditiX

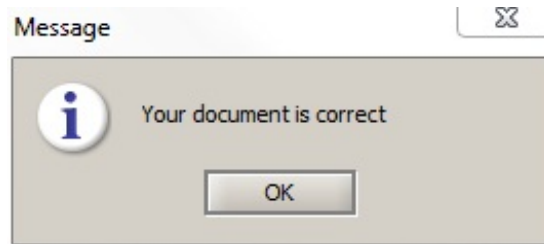
Un document bien formé nécessite un élément racine.

Il manque donc un élément racine. Complétons tout de suite notre document avec les éléments suivant :

Code : XML

```
<repertoire>  
  <personne>Robert</personne>  
  <personne>John</personne>  
</repertoire>
```

Lancer la vérification de votre document. Vous devriez avoir le message suivant à l'écran :



Félicitation, votre document est bien formé !

L'indentation

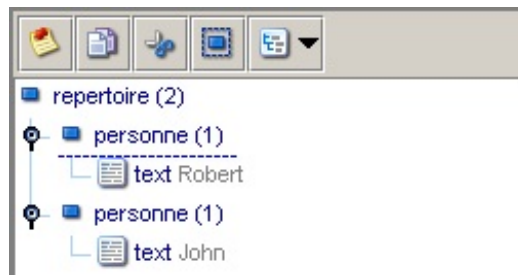
Il est possible de demander à Editix d'indenter automatiquement vos documents une fois écrit.

Pour se faire, sélectionnez dans la barre de menu **XML** puis **Format** et **Pretty format (default)** ou utilisez le raccourci clavier Ctrl + R.

Dans ce même menu, vous pouvez accéder aux paramètres concernant la tabulation.

L'arborescence du document


Editix met à votre disposition un outil fort sympathique qui vous permet de visualiser l'arborescence du document en cours d'édition.



On sait ainsi que notre répertoire contient 2 personnes. La première s'appelle Robert et la seconde John.

Enregistrer votre document

Il est maintenant temps de clore ce chapitre en enregistrant votre document XML.

Pour se faire, vous pouvez cliquer sur l'icône , sélectionner dans la barre de menu **Save** puis **Save** ou encore utiliser le raccourci clavier Ctrl + S.

Dans la fenêtre de dialogue qui vient de s'ouvrir, choisissez l'emplacement où vous souhaitez stocker votre fichier XML, tapez son nom et cliquez sur **Enregistrer**.

Croyez-moi ou pas, mais si vous lisez ce message, c'est que vous maîtrisez les bases du langage XML !

La suite de ce tutoriel sera l'occasion de voir plusieurs concepts rattachés au langage XML, mais à cette étape du tutoriel, vous êtes déjà capable de faire beaucoup ! 😊

Avant de passer à la seconde partie, je vous propose de tester les connaissances acquises au cours de cette première partie grâce à un petit TP.

TP : structuration d'un répertoire

Voici donc le premier TP de ce tutoriel ! L'objectif de ses chapitres un peu particulier est de vous faire pratiquer. L'objectif est de vous permettre de restituer toute la théorie que vous avez lu au cours des chapitres précédents. C'est selon moi, indispensable pour s'assurer que vous avez bien compris ce qui est expliqué.

Dans ce premier TP, l'objectif est de vous montrer une utilisation concrète de structuration de données via XML.

L'énoncé

Le but de ce TP est de créer un document XML structurant les données d'un répertoire.

Votre répertoire doit comprendre au moins 2 personnes. Pour chaque personne on souhaite connaître les informations suivantes :

- son sexe (homme ou femme) ;
- son nom ;
- son prénom ;
- son adresse ;
- un ou plusieurs numéros de téléphone (téléphone portable, fixe, bureau, etc.) ;
- une ou plusieurs adresses e-mail (adresse personnelle, professionnelle, etc.).

Je ne vous donne aucune indication concernant le choix des balises, des attributs et de l'arborescence à choisir pour une raison très simple : lorsqu'on débute en XML, le choix des attributs, des balises et de l'arborescence est assez difficile.

Le but est vraiment de vous laisser chercher et vous pousser à vous poser les bonnes questions sur l'utilité d'une balise, d'un attribut, etc.

Une solution

Je vous fais part de ma solution possible.

Secret ([cliquez pour afficher](#))

Code : XML

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

<repertoire>
  <!-- John DOE -->
  <personne sexe="masculin">
    <nom>DOE</nom>
    <prenom>John</prenom>
    <adresse>
      <numero>7</numero>
      <voie type="impasse">impasse du chemin</voie>
      <codePostal>75015</codePostal>
      <ville>PARIS</ville>
      <pays>FRANCE</pays>
    </adresse>
    <telephones>
      <telephone type="fixe">01 02 03 04 05</telephone>
      <telephone type="portable">06 07 08 09 10</telephone>
    </telephones>
    <emails>
      <email type="personnel">john.doe@wanadoo.fr</email>
      <email
type="professionnel">john.doe@societe.com</email>
    </emails>
  </personne>

  <!-- Marie POPPINS -->
  <personne sexe="feminin">
    <nom>POPPINS</nom>
    <prenom>Marie</prenom>
    <adresse>
      <numero>28</numero>
```



```
<voie type="avenue">avenue de la république</voie>
<codePostal>13005</codePostal>
<ville>MARSEILLE</ville>
<pays>FRANCE</pays>
</adresse>
<telephones>
  <telephone type="professionnel">04 05 06 07
08</telephone>
</telephones>
<emails>
  <email type="professionnel">contact@poppins.fr</email>
</emails>
</personne>
</repertoire>
```

Quelques explications

Le sexe

Comme vous pouvez le constater, j'ai fait le choix de renseigner le sexe dans un attribut de la balise `<personne/>` et non d'en faire une balise à part entière.

En effet, cette information est (je pense) plus utile à l'ordinateur qui lira le document qu'à un humain. En effet, contrairement à une machine, nous avons la capacité de déduire que John est un prénom masculin et Marie un prénom féminin. Cette information n'est donc pas cruciale pour les humains qui lisent le fichier.

L'adresse

Il est important que vos documents XML aient une arborescence logique. C'est pourquoi j'ai décidé de représenter l'adresse postale par une balise `<adresse />` qui contient les informations détaillées de l'adresse de la personne comme le numéro dans la rue, la voie, le pays, etc.

J'ai également fait le choix d'ajouter un attribut **type** dans la balise `<voie />`. Une nouvelle fois, cette attribut est destiné à être utilisé par une machine.

En effet, une machine qui traitera ce fichier, pourra facilement accéder au type de la voie sans avoir à récupérer le contenu de la balise `<voie/>` et tenter d'analyser s'il s'agit d'une impasse, d'une rue, d'une avenue, etc. C'est donc un gain de temps dans le cas d'un traitement des données.

Numéros de téléphone et adresses e-mails

Encore une fois, dans un soucis d'arborescence logique, j'ai décidé de créer les blocs `<telephones />` et `<emails />` qui contiennent respectivement l'ensemble des numéros de téléphone et des adresses e-mail.

Pour chacune des balises `<telephone/>` et `<email/>`, j'ai décidé d'y mettre un attribut **type**. Cette attribut permet de renseigner si l'adresse e-mail ou le numéro de téléphone est par exemple professionnel ou personnel.

Bien qu'indispensable aussi bien aux humains qu'aux machines, cette information est placée dans un attribut car ce n'est pas l'information principale que l'on souhaite transmettre. Ici l'information principale reste le numéro de téléphone ou l'adresse e-mail et non son type.



Si vous avez des questions ou si vous voulez faire valider votre solution, n'hésitez pas à passer par le forum de ce site internet ! 😊

Partie 2 : Créez des définitions pour vos documents XML

Maintenant que vous connaissez le XML, je vous propose de passer à la suite ! Dans cette seconde partie, nous verrons ensemble comment définir des structures précises pour nos documents. Ça sera l'occasion de découvrir 2 nouvelles technologies : les DTD et les Schémas XML.

Introduction aux définitions et aux DTD

Après avoir découvert le XML dans une première partie, nous n'allons pas nous arrêter là ! En effet, pour être tout à fait honnête avec vous, il très rare que les documents XML soient utilisés seuls. Ils sont généralement accompagnés d'un second fichier qui permet de lui définir une structure strict à avoir, c'est ce qu'on appelle un fichier de définition.

Dans ce premier chapitre, je vous propose de revenir un peu plus profondément sur ce qu'est une définition et pourquoi c'est une notion importante. Finalement, nous étudierons également l'une des deux technologies permettant d'écrire des définitions : les DTD.

Les éléments

La syntaxe

Pour définir les règles portant sur les **balises**, on utilise le mot clef **ELEMENT**.

Code : XML

```
<!ELEMENT balise (contenu)>
```

Une règle peut donc se découper en 3 mots clefs : **ELEMENT**, **balise** et **contenu**.

Retour sur la balise

Le mot-clef **balise** est à remplacer par le nom de la balise à laquelle vous souhaitez appliquer la règle.

Pour exemple, reprenons une balise du TP de la partie 1 :

Code : XML

```
<nom>DOE</nom>
```

On écrira alors :

Code : XML

```
<!ELEMENT nom (contenu)>
```

Retour sur le contenu

Cet emplacement a pour vocation de décrire ce que doit contenir la balise : est-ce une autre balise ou est-ce une valeur ?

Cas d'une balise en contenant une autre

Par exemple, regardons la règle suivante :

Code : XML

```
<!ELEMENT personne (nom)>
<!-- suite de la DTD -->
```

Cette règle signifie que la balise **<personne />** contient la balise **<nom />**.

Le document XML respectant cette règle ressemble donc à cela :

Code : XML

```
<personne>
  <nom>John DOE</nom>
</personne>
```



Nous n'avons défini aucune règle pour la balise **<nom/>**. Le document n'est en conséquence pas valide. **En effet, dans une DTD, il est impératif de décrire tout le document sans exception.** Des balises qui n'apparaissent pas dans la DTD ne peuvent pas être utilisées dans le document XML.

Cas d'une balise contenant une valeur

Dans le cas où notre balise contient une **valeur simple**, on utilisera le mot clef **#PCDATA**

Une valeur simple désigne par exemple une chaîne de caractères, un entier, un nombre décimal, un caractère, etc.

En se basant sur l'exemple précédent :

Code : XML

```
<personne>
  <nom>John DOE</nom>
</personne>
```

Nous avons déjà défini une règle pour la balise **<personne/>** :

Code : XML

```
<!ELEMENT personne (nom)>
```

Nous pouvons maintenant compléter notre DTD en ajoutant une règle pour la balise **<nom/>**. Par exemple, si l'on souhaite que cette balise contienne une valeur simple, on écrira :

Code : XML

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

Au final, la DTD de notre document XML est donc la suivante :

Code : XML

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

Cas d'une balise vide

Il est également possible d'indiquer qu'une balise ne contient rien grâce au mot clef **EMPTY**.

Prenons les règles suivantes :

Code : XML

```
<!ELEMENT personne (nom)>
<!ELEMENT nom EMPTY>
```

Le document XML répondant à la définition DTD précédente est le suivant :

Code : XML

```
<personne>
  <nom />
</personne>
```



A noter que lors de l'utilisation du mot clef **EMPTY**, l'usage des parenthèses n'est pas obligatoire !

Cas d'une balise contenant n'importe quoi

Il nous reste un cas à voir : celui d'une balise qui peut contenir n'importe quoi. C'est à dire que notre balise peut contenir une autre balise, une valeur simple ou être vide. Dans ce cas, on utilise le mot clef **ANY**.

Prenons la règle suivante :

Code : XML

```
<!ELEMENT personne (nom)>
<!ELEMENT nom ANY>
```

Les documents XML suivants sont bien valides :

Code : XML

```
<!-- valeur simple -->
<personne>
  <nom>John DOE</nom>
</personne>

<!-- vide -->
<personne>
  <nom />
</personne>
```



Bien que le mot clef **ANY** existe, il est souvent déconseillé de l'utiliser afin de restreindre le plus possible la liberté de rédaction du document XML.



Comme pour le mot clef **EMPTY**, l'usage des parenthèses n'est pas obligatoire pour le mot clef **ANY**!

Structurer le contenu des balises

Nous allons voir maintenant des syntaxes permettant d'apporter un peu de généricité aux définitions DTD.

Par exemple, un répertoire contient généralement un nombre variable de personnes, il faut donc permettre à un document XML d'être valide quelque soit le nombre de personnes qu'il contient.

La séquence

Une séquence permet de décrire l'enchaînement imposé des balises. Il suffit d'indiquer le nom des balises en les séparant par des virgules.

Code : XML

```
<!ELEMENT balise (balise2, balise3, balise4, balise5, etc.)>
```

Prenons l'exemple suivant :

Code : XML

```
<!ELEMENT personne (nom, prenom, age)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

Cette définition impose que la balise **<personne />** contienne obligatoirement les balises **<nom />**, **<prenom />** et **<age />** dans cet ordre.

Regardons alors la validité des documents XML qui suivent:

Code : XML

```
<!-- valide -->
<personne>
  <nom>DOE</nom>
  <prenom>John</prenom>
  <age>24</age>
</personne>

<!-- invalide -->
<!-- les balises ne sont pas dans le bon ordre -->
<personne>
  <prenom>John</prenom>
  <nom>DOE</nom>
  <age>24</age>
</personne>

<!-- invalide -->
<!-- il manque une balise -->
<personne>
  <prenom>John</prenom>
  <age>24</age>
```

```

</personne>

<!-- invalide -->
<!-- il y a une balise en trop qui plus est n'est pas déclarée -->
<personne>
  <nom>DOE</nom>
  <prenom>John</prenom>
  <age>24</age>
  <date>12/12/2012</date>
</personne>

```

La liste de choix

Une liste de choix permet de dire qu'une balise contient l'une des balises décrites. Il suffit d'indiquer le nom des balises en les séparant par une **barre verticale**.

Code : XML

```
<!ELEMENT balise (balise2 | balise3 | balise4 | balise5 | etc.)>
```

Prenons l'exemple suivant :

Code : XML

```

<!ELEMENT personne (nom | prenom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>

```

Cette définition impose que la balise **<personne />** contienne obligatoirement la balise **<nom />** ou la balise **<prenom />**.

Regardons alors la validité des documents XML ci-dessous :

Code : XML

```

<!-- valide -->
<personne>
  <nom>DOE</nom>
</personne>

<!-- valide -->
<personne>
  <prenom>John</prenom>
</personne>

<!-- invalide -->
<!-- les 2 balises prenom et nom ne peuvent pas être présentes en même temps. -->
<personne>
  <prenom>John</prenom>
  <nom>DOE</nom>
</personne>

<!-- invalide -->
<!-- il manque une balise -->
<personne />

```

La balise optionnelle

Une balise peut-être optionnelle. Pour indiquer qu'une balise est optionnelle, on fait suivre son nom par un **point d'interrogation**.

Code : XML

```
<!ELEMENT balise (balise2, balise3?, balise4)>
```

Prenons l'exemple suivant :

Code : XML

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

Cette définition impose que la balise **<personne />** contienne obligatoirement la balise **<nom />** puis éventuellement **<prenom />**.

Regardons alors la validité de ces documents XML :

Code : XML

```
<!-- valide -->
<personne>
  <nom>DOE</nom>
</personne>

<!-- valide -->
<personne>
  <nom>DOE</nom>
  <prenom>John</prenom>
</personne>

<!-- invalide -->
<!-- l'ordre des balises n'est pas respecté -->
<personne>
  <prenom>John</prenom>
  <nom>DOE</nom>
</personne>
```

La balise répétée optionnelle

Une balise peut-être répétée plusieurs fois bien qu'optionnelle. Pour indiquer une telle balise on fait suivre son nom par une **étoile**.

Code : XML

```
<!ELEMENT balise (balise2, balise3*, balise4)>
```

Soit l'ensemble de règles suivant :

Code : XML

```

<!ELEMENT repertoire (personne*)>
<!ELEMENT personne (nom, prenom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>

```

Cette définition impose que la balise **<repertoire />** contienne entre 0 et une infinité de fois la balise **<personne />**. La balise **<personne />** quant à elle doit obligatoirement contenir les balises **<nom />** et **<prenom />** dans cet ordre.

Regardons alors la validité des documents XML :

Code : XML

```

<!-- valide -->
<repertoire>
  <personne>
    <nom>DOE</nom>
    <prenom>John</prenom>
  </personne>
  <personne>
    <nom>POPPINS</nom>
    <prenom>Marie</prenom>
  </personne>
</repertoire>

<!-- valide -->
<repertoire>
  <personne>
    <nom>DOE</nom>
    <prenom>John</prenom>
  </personne>
</repertoire>

<!-- valide -->
<repertoire />

<!-- invalide -->
<!-- il manque la balise prenom dans la seconde balise personne-->
<repertoire>
  <personne>
    <nom>DOE</nom>
    <prenom>John</prenom>
  </personne>
  <personne>
    <nom>POPPINS</nom>
  </personne>
</repertoire>

```

La balise répétée

Une balise peut-être répétée plusieurs fois. Pour indiquer une telle balise on fait suivre son nom par un **plus**.

Code : XML

```

<!ELEMENT balise (balise2, balise3+, balise4)>

```

Prenons l'exemple suivant :

Code : XML


```
<!ELEMENT repertoire (personne+)>
<!ELEMENT personne (nom, prenom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
```

Cette définition impose que la balise **<repertoire />** contienne **au minimum une fois** la balise **<personne />**. La balise **<personne />** quant à elle doit obligatoirement contenir les balises **<nom />** et **<prenom />** dans cet ordre.

Regardons alors la validité des documents XML suivants :

Code : XML

```
<!-- valide -->
<repertoire>
  <personne>
    <nom>DOE</nom>
    <prenom>John</prenom>
  </personne>
  <personne>
    <nom>POPPINS</nom>
    <prenom>Marie</prenom>
  </personne>
</repertoire>

<!-- valide -->
<repertoire>
  <personne>
    <nom>DOE</nom>
    <prenom>John</prenom>
  </personne>
</repertoire>

<!-- invalide -->
<!-- la balise personne doit être présente au moins une fois-->
<repertoire />
```

En résumé

Nous venons de voir dans ce chapitre de nombreuses nouveautés, à savoir les prémices d'une nouvelle technologie, une nouvelle syntaxe et plein de mots clefs à retenir !

Je suis conscient que ce n'est pas facile à digérer, c'est pourquoi je vous encourage à relire plusieurs fois ce chapitre avant de passer à la suite afin que vous soyez certains d'avoir tout compris !

DTD : les attributs et les entités

Dans le chapitre précédent, nous avons vu comment décrire les balises de nos documents XML. Mais rappelez vous, une balise peut contenir ce qu'on appelle des attributs. C'est ce que nous allons voir au cours de ce chapitre.

Nous reviendrons également sur une notion dont je n'ai pas encore parlé : les entités. Je laisse un peu de suspense autour de la définition de cette notion et son utilisation. 😊

Les attributs

Dans le chapitre précédent, nous avons vu la syntaxe pour définir des règles sur les balises de nos documents XML. Vous allez voir que le principe est le même pour définir des règles à nos attributs.

La syntaxe

Pour indiquer que notre règle porte sur un **attribut**, on utilise le mot clef **ATTLIST**. On utilise alors la syntaxe suivante :

Code : XML

```
<!ATTLIST balise attribut type mode>
```

Une règle peut donc se découper en 5 mots clefs : **ATTLIST**, **balise**, **attribut**, **type** et **mode**.

Retour sur la balise et l'attribut

Il n'est nécessaire de s'attarder trop longtemps sur le sujet, il suffit simplement d'écrire le nom de la balise et de l'attribut concerné par la règle.

Par exemple, reprenons une balise du TP de la partie 1 :

Code : XML

```
<personne sexe="masculin" />
```

On écrira alors :

Code : XML

```
<!ATTLIST personne sexe type mode>
```

Retour sur le type

Cet emplacement a pour vocation de décrire le **type** de l'attribut. Est-ce une valeur bien précise ? du texte ? un identifiant ?

Cas d'un attribut ayant pour type la liste des valeurs possibles

Nous allons étudier ici le cas d'un attribut ayant pour type une liste de valeurs. Les différentes valeurs possibles pour l'attribut sont séparées par une **barre verticale**.

Code : XML

```
<!ATTLIST balise attribut (valeur 1 | valeur 2 | valeur 3 | etc.)
```

```
mode>
```

Reprenons une nouvelle fois la balise `<personne />`. Nous avons vu que cette balise possède un attribut `sexe`. Nous allons ici imposer la valeur que peut prendre cet attribut : soit **masculin**, soit **féminin**.

Voici ce à quoi la règle portant sur l'attribut dans notre DTD doit ressembler :

Code : XML

```
<!ATTLIST personne sexe (masculin|féminin) mode>
```

Quelques exemple de documents XML possibles :

Code : XML

```
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- invalide -->
<personne sexe="autre" />
```

Cas d'un attribut ayant pour type du texte non "parsé"

Derrière le terme **"texte non "parsé"** se cache en fait la possibilité de mettre ce que l'on veut comme valeur : un nombre, une lettre, une chaîne de caractères, etc. Il s'agit de données qui ne seront pas analysées par le "parseur" au moment de la validation et/ou l'exploitation de votre document XML.

Dans le cas où notre attribut contient du **texte non "parsé"**, on utilise le mot clef **CDATA**.

Code : XML

```
<!ATTLIST balise attribut CDATA mode>
```

Soit la règle suivante :

Code : XML

```
<!ATTLIST personne sexe CDATA mode>
```

Notre document XML répondant à cette règle peut ressembler à cela :

Code : XML

```
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />
```

```
<!-- valide -->
<personne sexe="autre" />

<!-- valide -->
<personne sexe="12" />
```

Cas d'un attribut ayant pour type un identifiant unique

Il est tout à fait possible de vouloir qu'une balise possède un attribut permettant de l'identifier de manière unique.

Prenons par exemple l'exemple d'une course à pied. Dans le classement de la course, il y aura un **unique** vainqueur, un **unique** second et un **unique** troisième.

Pour indiquer que la valeur de l'attribut est unique, on utilise le mot clef **ID** comme **ID**entifiant.

Code : XML

```
<!ATTLIST balise attribut ID mode>
```

Prenons par exemple la règle suivante :

Code : XML

```
<!ATTLIST personne position ID mode>
```

Voici quelques exemples de documents XML :

Code : XML

```
<!-- valide -->
<personne position="POS-1" />
<personne position="POS-2" />
<personne position="POS-3" />

<!-- invalide -->
<personne position="POS-1" />
<personne position="POS-1" />
<personne position="POS-2" />
```

Cas d'un attribut ayant pour type une référence à un identifiant unique

Il est tout à fait possible que dans votre document, un de vos attributs fasse référence à un identifiant. Cela permet souvent de ne pas écrire 100 fois les mêmes informations.

Par exemple, votre document XML peut vous servir à représenter des liens de parenté entre des personnes. Grâce aux références, nous n'allons pas devoir imbriquer des balises XML dans tous les sens pour tenter de représenter le père d'une personne ou le fils d'une personne.

Pour faire référence à un identifiant unique, on utilise le mot clef **IDREF**.

Prenons par exemple la règle suivante :

Code : XML

```
<!ATTLIST father id ID mode >
<!ATTLIST child id ID mode
father IDREF mode
>
```

Cette règle signifie que la balise **personne** a 2 attributs : **id** qui est l'identifiant unique de la personne et **father** qui fait référence une autre personne.

Illustrons immédiatement avec un exemple XML :

Code : XML

```
<!-- valide -->
<father id="PER-1" />
<child id="PER-2" father="PER-1" />

<!-- invalide -->
<!-- l'identifiant PER-0 n'apparaît nulle part -->
<father id="PER-1" />
<child id="PER-2" father="PER-0" />
```

Dans cet exemple, la personne **PER-2** a pour père la personne **PER-1**. On matérialise bien le lien entre ces 2 personnes.

Retour sur le mode

Cet emplacement permet de donner une information supplémentaire sur l'attribut comme par exemple une indication sur son obligation ou sa valeur.

Cas d'un attribut obligatoire

Lorsqu'on souhaite qu'un attribut soit obligatoirement renseigné, on utilise le mot clef **#REQUIRED**.

Par exemple, si l'on souhaite que le sexe d'une personne soit renseigné, on utilisera la règle suivante :

Code : XML

```
<!ATTLIST personne sexe (masculin|féminin) #REQUIRED>
```

Voici alors quelques exemple de documents XML possibles :

Code : XML

```
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- invalide -->
<personne />
```

Cas d'un attribut optionnel

Si au contraire on souhaite indiquer qu'un attribut n'est pas obligatoire, on utilise le mot clef **#IMPLIED**.

Si l'on reprend l'exemple précédent, on peut indiquer qu'il n'est pas obligatoire de renseigner le sexe d'une personne via la règle suivante :

Code : XML

```
<!ATTLIST personne sexe CDATA #IMPLIED>
```

Voici alors quelques exemple de documents XML possibles :

Code : XML

```
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- valide -->
<personne sexe="15" />

<!-- valide -->
<personne />
```

Cas d'une valeur par défaut

Il est également possible d'indiquer une valeur par défaut pour un attribut. Il suffit tout simplement d'écrire cette valeur en dur dans la règle.

Par exemple, il est possible d'indiquer qu'une personne dont l'attribut **sexe** n'est pas renseigné est un homme par défaut grâce à la règle suivante :

Code : XML

```
<!ATTLIST personne sexe CDATA "masculin">
```

Voici alors quelques exemple de documents XML possibles :

Code : XML

```
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- valide -->
<!-- l'attribut sexe vaut "masculin" -->
<personne />
```

Cas d'une constante

Finalement, il est possible de rendre obligatoire un attribut et de fixer sa valeur grâce au mot clef **#FIXED** suivi de la dite valeur.

Cette situation peut par exemple se rencontrer lorsqu'on souhaite travailler dans une devise bien précise et qu'on souhaite qu'elle apparaisse dans le document.

Par exemple, la règle suivante permet d'indiquer que la devise doit obligatoirement apparaître et a pour seule valeur possible l'euro.

Code : XML

```
<!ATTLIST objet devise CDATA #FIXED "Euro">
```

Voici alors quelques exemple de documents XML possibles :

Code : XML

```
<!-- valide -->
<objet devise="Euro" />

<!-- invalide -->
<objet devise="Dollar" />

<!-- invalide -->
<objet />
```

Les entités

Une autre notion assez importante à voir lorsqu'on parle de DTD est la notion d'entité.

Définition

Une entité peut-être considérée comme un alias permettant de réutiliser des informations au sein du document XML ou de la définition DTD.

Au cours de ce chapitre, nous reviendrons sur les 3 types d'entités : les entités générales, les entités paramètres et les entités externes.

Les entités générales

Définition

Les entités générales sont les entités les plus simples. Elles permettent d'associer un alias à une information afin de l'utiliser dans le document XML.

La syntaxe

Voyons tout de suite la syntaxe d'une entité générale :

Code : XML

```
<!ENTITY nom "valeur">
```

Pour utiliser une entité générale dans notre document XML, il suffit d'utiliser la syntaxe suivante :

Code : XML

```
&nom;
```

Afin d'illustrer un peu plus clairement mes propos, voyons tout de suite un exemple :

Code : XML

```
<!ENTITY samsung "Samsung">
<!ENTITY apple "Apple">

<telephone>
  <marque>&samsung;</marque>
  <modele>Galaxy S3</modele>
</telephone>
<telephone>
  <marque>&apple;</marque>
  <modele>iPhone 4</modele>
</telephone>
```

Au moment de son interprétation, les références aux entités seront remplacées par leurs valeurs respectives, ce qui donne une fois interprété :

Code : XML

```
<telephone>
  <marque>Samsung</marque>
  <modele>Galaxy S3</modele>
</telephone>
<telephone>
  <marque>Apple</marque>
  <modele>iPhone 4</modele>
</telephone>
```

Les entités paramètres

Définition

Contrairement aux entités générales qui apparaissent dans les documents XML, les entités paramètres n'apparaissent que dans les définitions DTD. Elles permettent d'associer un alias à une partie de la déclaration de la DTD.

La syntaxe

Voyons tout de suite la syntaxe d'une entité paramètre :

Code : XML

```
<!ENTITY % nom "valeur">
```

Pour utiliser une entité paramètre dans notre DTD, il suffit d'utiliser la syntaxe suivante :

Code : XML


```
%nom;
```

Prenons par exemple cet exemple où des téléphones ont pour attribut une marque :

Code : XML

```
<telephone marque="Samsung" />  
<telephone marque="Apple" />
```

Normalement, pour indiquer que l'attribut `marque` de la balise `<telephone/>` est obligatoire et qu'il doit contenir la valeur Samsung ou Apple, nous devons écrire la règle suivante :

Code : XML

```
<!ATTLIST telephone marque (Samsung|Apple) #REQUIRED>
```

A l'aide d'une entité paramètre, cette même règle s'écrit de la façon suivante :

Code : XML

```
<!ENTITY % listeMarques "marque (Samsung|Apple) #REQUIRED">  
<!ATTLIST telephone %listeMarques; >
```

Encore une fois, au moment de son interprétation, les références aux entités seront remplacées par leurs valeurs respectives.

Les entités externes

Définition

Il existe en réalité 2 types d'entités externes : les analysées et les non analysées. Dans le cadre de ce cours, nous nous limiterons aux entités externes analysées.

Les entités externes analysées ont sensiblement le même rôle que les entités générales, c'est à dire qu'elles permettent d'associer un alias à une information afin de l'utiliser dans le document XML. Mais dans le cas des entités externes analysées, les informations sont stockées dans un fichier séparés.

La syntaxe

Voyons tout de suite la syntaxe d'une entité externe :

Code : XML

```
<!ENTITY nom SYSTEM "URI">
```

Pour utiliser une entité externe dans notre XML, il suffit d'utiliser la syntaxe suivante :

Code : XML

```
&nom;
```

Si l'on reprend notre premier exemple, voici ce que cela donne :

Code : XML

```
<!ENTITY samsung SYSTEM "samsung.xml">
<!ENTITY apple SYSTEM "apple.xml">

<telephone>
  &samsung;
  <modele>Galaxy S3</modele>
</telephone>
<telephone>
  &apple;
  <modele>iPhone 4</modele>
</telephone>
```

Le contenu des fichiers **samsung.xml** et **apple.xml** sera par exemple le suivant :

Code : XML

```
<!-- Contenu du fichier samsung.xml -->
<marque>Samsung</marque>

<!-- Contenu du fichier apple.xml -->
<marque>Apple</marque>
```

Au moment de son interprétation, les références aux entités seront remplacées par leurs valeurs respectives, ce qui donne une fois interprété :

Code : XML

```
<telephone>
  <marque>Samsung</marque>
  <modele>Galaxy S3</modele>
</telephone>
<telephone>
  <marque>Apple</marque>
  <modele>iPhone 4</modele>
</telephone>
```

En résumé

Dans ce second chapitre, nous venons de voir comment définir les attributs de vos balises XML. Nous avons également découvert une nouvelle notion : les entités. Elles nous permettent de jouer les fainéants en réutilisant des éléments qui reviennent souvent dans nos documents.

DTD : où les écrire ?

Nous venons donc d'étudier au cours des derniers chapitres tout ce qu'il faut savoir ou presque sur les DTD. En effet, vous ignorez encore une information importante sur les DTD vous permettant de passer de la théorie à la pratique : où les écrire.

Ca sera également l'occasion de vous révéler qu'il existe en réalité plusieurs sortes de DTD.

Les DTD internes

Comme je vous l'ai déjà précisé dans le premier chapitre de cette seconde partie, on distingue 2 types de DTD : les internes et les externes.

Commençons par étudier les internes !

Définition

Une DTD interne est une DTD qui est écrite dans le même fichier que le document XML. Elle est généralement spécifique au document XML dans lequel elle est écrite.

La Syntaxe

Une DTD interne s'écrit dans ce qu'on appelle le **DOCTYPE**. On le place sous le prologue du document et au dessus du contenu XML.

Voyons plus précisément la syntaxe :

Code : XML

```
<!DOCTYPE racine [ ]>
```

La DTD interne est ensuite écrite entre les []. Dans ce **DOCTYPE**, le mot **racine** doit être remplacé par le nom de la balise qui forme la racine du document XML.

Illustrons avec un exemple

Afin que tout cela vous paraisse moins abstrait, je vous propose de voir un petit exemple.

Prenons par exemple l'énoncé suivant :

Citation : Énoncé

Une boutique possède plusieurs téléphones. Chaque téléphone est d'une certaine marque et d'un certain modèle représenté par une chaîne de caractère.

Un document XML répondant à cet énoncé est par exemple le suivant :

Code : XML

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

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
```

```
<modele>iPhone 4</modele>
</telephone>

<telephone>
  <marque>Nokia</marque>
  <modele>Lumia 800</modele>
</telephone>
</boutique>
```

La définition DTD est la suivante :

Code : XML

```
<!ELEMENT boutique (telephone*)>
<!ELEMENT telephone (marque, modele)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT modele (#PCDATA)>
```

Le document XML complet avec la DTD interne sera par conséquent le suivant :

Code : XML

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

<!DOCTYPE boutique [
<!ELEMENT boutique (telephone*)>
  <!ELEMENT telephone (marque, modele)>
    <!ELEMENT marque (#PCDATA)>
    <!ELEMENT modele (#PCDATA)>
] >

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

Maintenant que vous savez ce qu'est une DTD interne, passons à la DTD externe !

Les DTD externes

Définition

Une DTD externe est une DTD qui est écrite dans un autre document que le document XML. Si elle est écrite dans un autre document, c'est que souvent, elle est commune à plusieurs documents XML qui l'exploitent.

De manière générale, afin de bien séparer le contenu XML de sa définition DTD, on prendra l'habitude de séparer dans plusieurs

fichiers.

Un fichier contenant uniquement une DTD porte l'extension **.dtd**.

La Syntaxe

L'étude de la syntaxe d'une DTD externe est l'occasion de vous révéler qu'il existe en réalité 2 types : les DTD externes **PUBLIC** et les DTD externes **SYSTEM**.

Dans les 2 cas et comme pour une DTD interne, c'est dans le **DOCTYPE** que cela se passe.

Les DTD externes PUBLIC

Les DTD externes PUBLIC sont généralement utilisées lorsque la DTD est une norme. C'est par exemple le cas dans les documents XHTML 1.0.

La syntaxe est la suivante :

Code : XML

```
<!DOCTYPE racine PUBLIC "identifiant" "url">
```

Si on l'applique à un document XHTML, on obtient alors le **DOCTYPE** suivant :

Code : XML

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

Pour être honnête, nous n'allons jamais utiliser les DTD externes PUBLIC dans ce tutoriel, c'est pourquoi je vous propose de passer immédiatement aux DTD externes SYSTEM.

Les DTD externes SYSTEM

Une DTD interne SYSTEM permet d'indiquer au document XML l'adresse du document DTD. Cette adresse peut-être relative ou absolue.

Voilà plus précisément la syntaxe :

Code : XML

```
<!DOCTYPE racine SYSTEM "URI">
```

Afin d'illustrer mes propos, je vous propose de reprendre l'exemple de la boutique de téléphone que j'ai utilisé dans la partie sur la DTD interne.

Voilà un rappel de l'énoncé :

Citation : Énoncé

Une boutique possède plusieurs téléphones. Chaque téléphone est d'une certaine marque et d'un certain modèle, tous les 2 représentés par une chaîne de caractère.

Pour rappel, voici le fichier XML :

Code : XML

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

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

Si la DTD ne change pas, elle doit cependant être placée dans un fichier à part, par exemple le fichier **doc1.dtd**. Voici son contenu :

Code : XML

```
<!ELEMENT boutique (telephone*)>
<!ELEMENT telephone (marque, modele)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT modele (#PCDATA)>
```

Le document XML complet avec la DTD externe sera alors le suivant (on part ici du principe que le fichier XML et DTD sont stockés au même endroit) :

Code : XML

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

<!DOCTYPE boutique SYSTEM "doc1.dtd">

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

Retour sur le prologue

Dans la partie précédente de ce tutoriel, voici ce que je vous avais dit à propos du fait que nos documents XML soient autonomes ou non :

Citation : Wapiti89

La dernière information présente dans le prologue est `standalone="yes"`.

Cette information permet de savoir si votre document XML est autonome ou si un autre document lui est rattaché.

Il est encore un peu tôt pour vous en dire plus. Nous reviendrons sur cette notion dans la partie 2 du tutoriel. Pour le moment acceptez le fait que nos documents sont tous autonomes.

Il est maintenant temps de lever le mystère !

Dans le cas d'une DTD externe, nos documents XML ne sont plus autonomes, en effet, ils font références à un autre fichier qui fournit la DTD. Afin que le document contenant la DTD soit bien pris en compte, nous devons l'indiquer en passant simplement la valeur de l'attribut `standalone` à **"no"**.

Voici ce que cela donne :

Code : XML

```
<?xml version = "1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE boutique SYSTEM "doc1.dtd">

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

Un exemple avec EditiX

Afin de terminer ce chapitre, je vous propose de voir ensemble comment écrire une DTD externe SYSTEM avec EditiX.

Pour faire simple, je vous propose de garder l'exemple précédent de la boutique de téléphone.

Création du document XML

La création du document XML n'a rien de bien compliqué puisque nous l'avons déjà vu ensemble dans la partie précédente.

Pour ceux qui ne s'en souviennent pas, vous pouvez y jeter un coup d'œil [ici](#).

Voici le document que vous devez écrire :

Code : XML

```
<?xml version = "1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE boutique SYSTEM "boutique.dtd">

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

Si vous essayez de lancer la vérification du document, vous devriez normalement avoir le message d'erreur suivant :

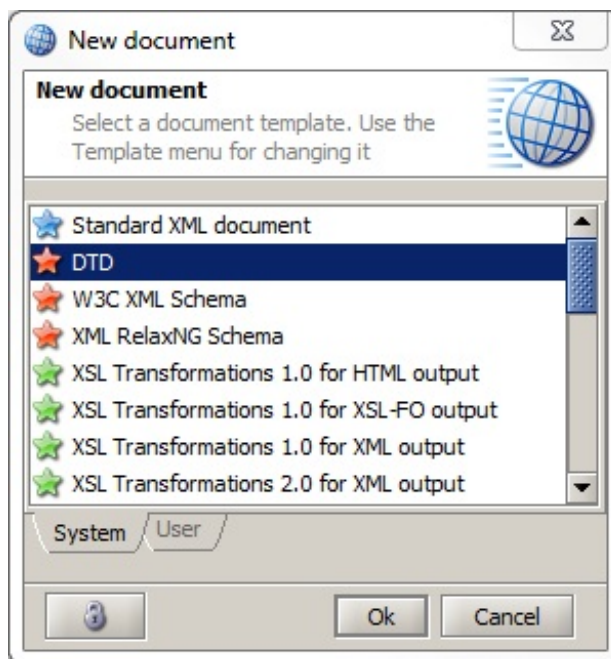


Ce message est pour le moment complètement normal puisque nous n'avons pas encore créé notre document DTD.

Création du document DTD

Pour créer un nouveau document, vous pouvez cliquer sur l'icône , sélectionner dans la barre de menu **File** puis **New** ou encore utiliser le raccourci clavier Ctrl + N.

Dans la liste qui s'affiche, sélectionnez **DTD**.



Votre document DTD n'est normalement pas vierge. Voici ce que vous devriez avoir :

Code : XML

```
<!-- DTD created at Wed Sep 12 14:49:47 CEST 2012 with EditiX.
Please insert an encoding attribute header for converting any DTD -
-->

<!ELEMENT tag (#PCDATA)>
<!ATTLIST tag attribute CDATA #REQUIRED>
```


Remplacez le contenu par notre véritable DTD :

Code : XML

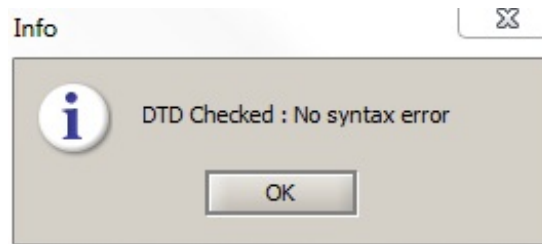
```
<!ELEMENT boutique (telephone*)>
<!ELEMENT telephone (marque, modele)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT modele (#PCDATA)>
```

Enregistrez ensuite votre document avec le nom **boutique.dtd** au même endroit que votre document XML.

Vérification de la DTD


Vous pouvez vérifier que votre DTD n'a pas d'erreur de syntaxe en cliquant sur l'icône , en sélectionnant dans la barre de menu **DTD/Schema** puis **Check this DTD** ou encore en utilisant le raccourci clavier Ctrl + K.

Vous devriez normalement avoir le message suivant :

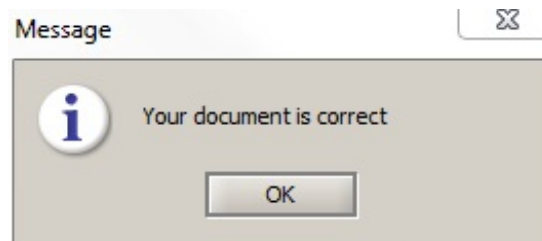


Vérification du document XML

Il est maintenant temps de vérifier que le document XML est **valide** !

Pour se faire, cliquez sur l'icône , sélectionnez dans la barre de menu **XML** puis **Check this document** ou encore en utilisant le raccourci clavier Ctrl + K.

Le message suivant doit normalement s'afficher :



En résumé

Nous venons donc de voir qu'il est possible d'écrire une DTD au sein même du fichier XML qu'elle définit. Cependant, nous privilégierons l'écriture de ces définitions dans un second fichier portant l'extension **.dtd**.

TP : définition DTD d'un répertoire

Votre apprentissage des DTD arrive donc à son terme et rien ne vaut un TP pour le conclure !

Pour ce TP, je vous propose de réaliser la définition DTD d'un répertoire téléphonique. Si en soit le sujet n'est pas extrêmement compliqué, il a le mérite de vous faire mettre en pratique toutes les notions vues jusque là dans cette seconde partie.

L'énoncé

Le but de ce TP est de créer la DTD du répertoire élaboré dans le premier TP.

Pour rappel, voici les informations que l'on souhaite connaître pour chaque personne :

- son sexe (homme ou femme) ;
- son nom ;
- son prénom ;
- son adresse ;
- un ou plusieurs numéros de téléphone (téléphone portable, fixe, bureau, etc.) ;
- une ou plusieurs adresses e-mail (adresse personnelle, professionnelle, etc.).

Voici le document XML que nous avons construit :

Code : XML

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

<repertoire>
  <!-- John DOE -->
  <personne sexe="masculin">
    <nom>DOE</nom>
    <prenom>John</prenom>
    <adresse>
      <numero>7</numero>
      <voie type="impasse">impasse du chemin</voie>
      <codePostal>75015</codePostal>
      <ville>PARIS</ville>
      <pays>FRANCE</pays>
    </adresse>
    <telephones>
      <telephone type="fixe">01 02 03 04 05</telephone>
      <telephone type="portable">06 07 08 09 10</telephone>
    </telephones>
    <emails>
      <email type="personnel">john.doe@wanadoo.fr</email>
      <email type="professionnel">john.doe@societe.com</email>
    </emails>
  </personne>

  <!-- Marie POPPINS -->
  <personne sexe="feminin">
    <nom>POPPINS</nom>
    <prenom>Marie</prenom>
    <adresse>
      <numero>28</numero>
      <voie type="avenue">avenue de la république</voie>
      <codePostal>13005</codePostal>
      <ville>MARSEILLE</ville>
      <pays>FRANCE</pays>
    </adresse>
    <telephones>
      <telephone type="professionnel">04 05 06 07
08</telephone>
    </telephones>
    <emails>
      <email type="professionnel">contact@poppins.fr</email>
```

```

    </emails>
  </personne>
</repertoire>

```

Une dernière consigne : la DTD doit être une DTD externe !

Une solution

Une fois de plus, je vous fais part de ma solution !

Le fichier XML avec le DOCTYPE :

Secret ([cliquez pour afficher](#))

Code : XML

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<!DOCTYPE repertoire SYSTEM "repertoire.dtd">

<repertoire>
  <!-- John DOE -->
  <personne sexe="masculin">
    <nom>DOE</nom>
    <prenom>John</prenom>
    <adresse>
      <numero>7</numero>
      <voie type="impasse">impasse du chemin</voie>
      <codePostal>75015</codePostal>
      <ville>PARIS</ville>
      <pays>FRANCE</pays>
    </adresse>
    <telephones>
      <telephone type="fixe">01 02 03 04 05</telephone>
      <telephone type="portable">06 07 08 09 10</telephone>
    </telephones>
    <emails>
      <email type="personnel">john.doe@wanadoo.fr</email>
      <email
type="professionnel">john.doe@societe.com</email>
      </emails>
    </personne>

    <!-- Marie POPPINS -->
    <personne sexe="feminin">
      <nom>POPPINS</nom>
      <prenom>Marie</prenom>
      <adresse>
        <numero>28</numero>
        <voie type="avenue">avenue de la république</voie>
        <codePostal>13005</codePostal>
        <ville>MARSEILLE</ville>
        <pays>FRANCE</pays>
      </adresse>
      <telephones>
        <telephone type="professionnel">04 05 06 07
08</telephone>
      </telephones>
      <emails>
        <email type="professionnel">contact@poppins.fr</email>
      </emails>
    </personne>
  </repertoire>

```

Le fichier DTD :

Secret (cliquez pour afficher)

Code : XML

```
<!-- Racine -->
<!ELEMENT repertoire (personne*)>

<!-- Personne -->
<!ELEMENT personne (nom, prenom, adresse, telephones, emails)>
<!ATTLIST personne sexe (masculin | feminin) #REQUIRED>

<!-- Nom et prénom -->
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>

<!-- Bloc adresse -->
<!ELEMENT adresse (numero, voie, codePostal, ville, pays)>
<!ELEMENT numero (#PCDATA)>

<!ELEMENT voie (#PCDATA)>
<!ATTLIST voie type CDATA #REQUIRED>

<!ELEMENT codePostal (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT pays (#PCDATA)>

<!-- Bloc téléphone -->
<!ELEMENT telephones (telephone+)>
<!ELEMENT telephone (#PCDATA)>
<!ATTLIST telephone type CDATA #REQUIRED>

<!-- Bloc email -->
<!ELEMENT emails (email+)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST email type CDATA #REQUIRED>
```

Un bref commentaire

Dans cette solution, je suis allé au plus simple en indiquant que pour les types de téléphones, d'e-mails et de voies, j'accepte toutes les chaînes de caractères. Libre à vous de créer de nouvelles règles si vous souhaitez que par exemple le choix du type de la voie ne soit possible qu'entre rue, avenue, impasse, etc.

Ce tutoriel est en cours d'écriture, n'hésitez pas à passer régulièrement pour voir les nouveautés ! 😊

Un grand merci à [Fumble](#) pour ses relectures, ses conseils et ses encouragements tout au long de la rédaction de ce cours.

Je souhaite également remercier [AnnaStretter](#) pour ses conseils afin de rendre ce cours plus pédagogique.