

# XML avancé : XPath et XSLT

Vous avez déjà entendu parler de XSL, XSLT, ou même de XPath. Lorsque les gens parlent de XSL, ils font parfois des confusions. Ceux sont des sujets compliqués à vrai dire, et il est temps de les maîtriser. Cette annexe fournit une rapide introduction à XSLT et XPath, mais si vous êtes attirés par ces technologies, nous vous recommandons d'acheter un livre à part ou de consulter les ressources additionnelles.

## QU'EST CE QUE XSL ?

XSL est un langage basé sur XML qui est devenu une recommandation du W3C à la fin de la dernière décennie. Il comprend trois sous-parties :

- XSLT : Un langage pour la transformation de documents XML.
- XPath : Un langage pour accéder et se référer à toute partie d'un document XML.
- XSL-FO : Un vocabulaire pour la sémantique de formatage.

W3C définit XSL ainsi : « XSL (Extensible Stylesheet Language) est un langage pour décrire les feuilles de style. Cela consiste en deux parties : un langage de transformation des documents XML, et un vocabulaire XML pour la spécification de la sémantique de formatage ».

Êtes-vous surpris que la définition elle-même relie XSL à CSS ? Eh bien, vous ne devriez pas l'être, parce le nom XSL inclut 'stylesheet'. Les origines de XSL sont dans CSS, XSL partage les mêmes fonctionnalités et est compatible CSS2 ; mais il y a quelques différences :

- XSL ajoute un langage de transformation appelé XSLT ; tandis que CSS vous permet de définir les types de polices de caractères, les fonds de pages, et les couleurs sur une page HTML. XSLT vous permet de transformer un document XML en une page (X) HTML, mais dans le même temps, il peut aussi transformer un document XML en un fichier au format texte.
- XSL ajoute des fonctions de styles avancées, les groupant sous une définition d'objets et d'attributs formatés.

Parce que nous avons parlé de CSS et XSL, il est normal de se poser des questions à leur sujet :

- Par quoi se fait la différence entre XSL et CSS ?
- Est-ce que XSL est à XML ce qu'est CSS pour HTML ?

Ces deux questions viennent naturellement, et il est préférable de leur apporter une réponse le plus tôt possible avant d'avoir de passer à la partie implémentation.

### **Par quoi se fait la différence entre XSL et CSS ?**

Cette question pourrait déjà avoir partiellement reçu une réponse dans les paragraphes précédents, mais nous allons essayer de rendre les choses plus transparentes. CSS utilise les balises prédéfinies du HTML pour formater un document, alors que XSL utilise les balises XML dans un document XML. Un autre aspect important est comment les feuilles de styles sont appliquées depuis la source de l'arborescence. Avec CSS, lorsque l'application des propriétés de formatage du sommet de l'arborescence est propagée à l'arbre suivant sa structure, cela a pour conséquence que les propriétés se reflètent de façons identiques dans l'arbre résultant de la transformation. Avec XSL, les choses sont différentes. Cela est dû au fait que nous avons un objet de type « arbre » (le document XSL, qui est lui-même un document XML) et nous pouvons appliquer les propriétés de formatage différemment, donc nous pourrions avoir un résultat très différent. L'héritage est obtenu par le formatage de l'objet « arbre » et non via le sommet de l'arborescence.

À côté de ces détails techniques, les implémentations de CSS (1 et 2) sont largement disponibles, ainsi que pour XSL, les problèmes sont réellement peu significatifs.

### **Est-ce que XSL est à XML ce qu'est CSS pour HTML ?**

CSS peut être utilisé pour un document HTML, mais il peut aussi être utilisé par un document XML. Lorsqu'il est utilisé dans des documents XML, il est fortement recommandé d'avoir une structure linéaire dans ceux-ci, ainsi la transformation demandée sera d'autant plus simple.

XSL est fait pour XML et cela va aussi très bien pendant de grosses manipulations sur le document XML désigné.

Comme vous pouvez le voir, c'est sûr que CSS et XSL vont de paires ensemble. Par exemple, vous pouvez utiliser XSL du côté serveur pour la simplification d'un document XML et CSS pour le formatage du côté client.

La plus communément, les deux coexisteront à l'avenir, parce que leurs domaines d'applications se croisent, ils ne sont pas inclus l'un dans l'autre. Ils sont tous deux très importants, vous avez besoin de bien les comprendre afin d'être capable de prendre les bonnes décisions pour vos projets.

Pour bien comprendre comment les choses évoluent, jetons un œil en arrière et voyons comment cela a démarré.

## Historique

XSL est apparu à cause d'un besoin d'avoir un langage pour gérer la définition des balises clientes de XML. En septembre 1998, Microsoft, Texcel et webMethods ont soumis une proposition au W3C sous le nom de langage de requête XML ou XQL. En Mai 1999, le W3C a décidé d'unifier les recherches dans cette direction sous *un noyau commun de modèles sémantiques* pour le requêtage, et le résultat de cela a été XSL pour la transformation ou XSLT.

Durant le développement de XSLT, une autre spécification de la famille XML était en cours : XPointer. XPointer utilise l'idée de balises ancrées dans le but de permettre l'adressage à des parties différentes du document.

Le besoin de sélection de différentes parties de documents XML, pour différents résultats, a résulté d'une sémantique et syntaxe commune, connu sous le nom de XPath. Cependant XPath est en théorie un sous-ensemble de XSLT, il peut être utilisé indépendamment. Aujourd'hui, XSLT, XPointer (avec XLink), et XQuery utilisent XPath pour sélectionner différentes parties dans un document XML.

Après avoir fait le tour d'horizon de XSL, il est temps pour nous d'avancer et de voir les spécificités de XSLT et XPath. Bien que XPath peut être vu comme un langage à part, nous commencerons avec lui.

## Installation

Ce tutorial utilisera le document XML suivant :

```
< ?xml version="1.0" encoding="ISO-8859-1"?>
<data>
  <params>
    <returned_page>1</returned_page>
    <total_pages>6</total_pages>
    <items_count>56</items_count>
    <previous_page></previous_page>
    <next_page>2</next_page>
  </params>
  <grid>
    <row>
      <product_id>1</product_id>
```

```

    <name>Santa Costume </name>
    <price>14.99</price>
    <on_promotion>1</on_promotion>
</row>
<row>
    <product_id>2</product_id>
    <name>Medieval Lady</name>
    <price>49.99</price>
    <on_promotion>1</on_promotion>
</row>
<row>
    <product_id>3</product_id>
    <name>Caveman</name>
    <price>12.99</price>
    <on_promotion>0</on_promotion>
</row>
<row>
    <product_id>4</product_id>
    <name>Costume Ghoul</name>
    <price>18.99</price>
    <on_promotion>0</on_promotion> </row>
<row>
    <product_id>5</product_id>
    <name>Ninja</name>
    <price>15.99</price>
    <on_promotion>0</on_promotion>
</row>
<row>
    <product_id>6</product_id>
    <name>Monk</name>
    <price>13.99</price>
    <on_promotion>0</on_promotion>
</row>
<row>
    <product_id>7</product_id>
    <name>Elvis Black Costume</name>
    <price>35.99</price>
    <on_promotion>0</on_promotion>
</row>
<row>
    <product_id>8</product_id>
    <name>Robin Hood</name>
    <price>18.99</price>
    <on_promotion>0</on_promotion>
</row>
<row>
    <product_id>9</product_id>
    <name>Pierot Clown</name>
    <price>22.99</price>
    <on_promotion>1</on_promotion>
</row>
<row>
    <product_id>10</product_id>
    <name>Austin Powers</name>
    <price>49.99</price>

```

```

    <on_promotion>0</on_promotion>
  </row>
</grid>
</data>

```

Le document XML contient une donnée extraite d'une impression de la grille de données AJAX d'un script côté serveur. Chaque produit a un identifiant (ID), un nom, un prix, et peut être en promotion.

## XPATH

XPath est un langage pour retrouver, sélectionner, et filtrer des informations depuis un document XML. Cette information peut être utilisée pour différentes tâches de traitement. Après avoir été une recommandation du W3C le 16 Novembre 1999, XPath a été utilisé dans différents langages basés sur XML.

XPath offre plus de 100 fonctions pour différentes tâches : manipulation des nœuds, fonctions de date et heure, fonctions sur les nombres, les fonctions sur les chaînes de caractères, et plus encore. Tout cela permet d'une manière simple d'exécuter les données depuis n'importe quel point du document XML.

Chaque document XML est modelé comme un arbre de nœuds. Il y a sept types de nœuds :

- Les nœuds racine.
- Les nœuds d'éléments.
- Les nœuds de texte.
- Les nœuds d'attributs.
- Les nœuds d'espace nom.
- Les nœuds de traitement des instructions.
- Les nœuds de commentaires.

Les types de base sont :

- Définition de nœuds (une collection non ordonnée de nœuds sans doublons).
- Booléen (vrai ou faux).
- Nombre (nombre à virgule flottante).
- Chaîne de caractères.

La construction syntaxique basique de XPATH est l'expression. L'expression est comparée à un objet qui peut avoir un des quatre types basiques que nous avons plus haut. Chaque expression est évaluée dans un contexte donné. Ce contexte est donné par le langage utilisant XPATH : XSLT, ou XPointer. Le contexte peut être un des suivants :

- Un nœud (contexte du nœud).
- Une paire d'entier positif non nul (contexte de position et contexte de taille).
- Un ensemble de variables liées aux données.

- Une fonction librairie.
- Un ensemble de déclarations d'espace de nom dans le champ de l'expression.

Les chemins de localisation sont les types d'expression les plus importants et c'est pour cela que nous avons besoin de les présenter en premier. Ils sélectionnent un ensemble de nœuds dans le contexte du nœud et contiennent aussi des expressions qui filtrent l'ensemble des nœuds qui ont été sélectionnés.

Il y a deux sortes de chemin de localisation :

- Les chemins absolus.
- Les chemins relatifs.

Les chemins relatifs consistent en plusieurs étapes de localisation séparées par des "/". Les chemins sont interprétés de gauche à droite. Chaque étape de localisation retourne un ensemble de nœuds qui est substantiellement utilisé pour la prochaine étape comme le contexte courant. Le chemin absolu contient le nœud racine ("/") suivi par un chemin relatif.

Un exemple de chemin absolu : /étape/étape/étape

Et un chemin relatif : étape/étape/étape

Une étape de localisation a trois sous-parties :

- *Un axe*, qui indique les trois relations entre les nœuds sélectionnés par l'étape de la localisation et le contexte du nœud.
- *Un test de nœud*, qui indique le nœud type, des nœuds sélectionnés par l'étape de la localisation.
- Zéro ou plus d'attributs qui utilisent arbitrairement des expressions pour redéfinir davantage un ensemble de nœuds sélectionné dans l'étape de la localisation.

La syntaxe pour l'étape de la localisation contient un axe suivi par un double deux-points, le nœud "testé", et zéro ou plus de pré-indications dans les crochets.

```
| Axe ::nœud testé[predicate]
    Ou :
```

```
| Axe ::noeud testé[predicate1][predicate2]
```

La table suivante montre les axes disponibles et leur description :

Tableau C.1 - Axe XPath

Axe	Description
Enfant	Contient les enfants du contexte de nœud
Descendant	Contient les descendants du contexte du nœud ; un descendant est un enfant ou un enfant d'un enfant et ainsi de suite, l'axe des descendants ne contient jamais d'attributs ou des nœuds d'espace de nom.
Parent	Contient le parent du contexte de nœud, s'il y en a un.
Ancestor	Contient les ancêtres du contexte de nœud ; les ancêtres du contexte de

	nœuds consistent en des parents de contexte de nœuds et de parents de parents et ainsi de suite ; l'axe d'ancêtre inclura toujours le nœud racine, à moins que le contexte de nœud est le nœud racine.
follo wing-sibli ng	Contient tous les frères ou sœurs suivants du nœud de contexte ; si le nœud de contexte est un nœud d'attribut ou le nœud d'espace de nom, l'axe frère ou sœur de suivant est vide.
precedi ng-sibli ng	Contient tous les frères ou sœurs précédant du nœud de contexte ; si le nœud de contexte est un nœud d'attribut ou un nœud d'espace de nom, l'axe précédant de frère ou sœur est vide.
Follo wing	Contient tous les nœuds dans le même document comme un nœud de contexte, qui sont après le nœud de contexte dans l'ordre du document, excluant n'importe quels descendants et excluant les attributs de nœuds et nœuds d'espace de nom.
Precedi ng	Contient tous nœuds dans le même document comme un nœud de contexte, qui sont avant le nœud de contexte dans l'ordre de document, excluant n'importe quels ancêtres et excluant les attributs de nœuds et nœuds d'espace de nom.
Attribute	Contient les attributs du nœud de contexte ; l'axe sera vide à moins que le nœud de contexte est un élément.
Namespace	Contient les nœuds d'espace de nom du nœud de contexte ; l'axe sera vide à moins que le nœud de contexte est un élément.
Sel f	Contient juste le nœud de contexte lui-même.
descendant-or-sel f	Contient le nœud de contexte et les descendants du nœud de contexte.
ancestor-or-sel f	Contient le nœud de contexte et les ancêtres du nœud de contexte ; ainsi l'axe d'ancêtre inclura toujours le nœud fondamental.

Les filtres d'attribut d'un ensemble de nœuds sont définis par un axe et produisent un nouvel ensemble de nœuds. Pour chaque nœud dans l'ensemble de nœuds désigné par l'axe, les attributs sont évalués avec le nœud considéré comme courant. Si l'attribut est évalué à VRAI, le nœud est inclus dans le nouvel ensemble de nœuds, autrement il n'est pas inclus.

Pour les chemins de localisation, nous pouvons utiliser une abréviation de la syntaxe, comme décrite dans le tableau dessous :

Tableau C.2 - Syntaxe abrégée de XPath

Abbreviated syntax	Description
Child	Contient les enfants du contexte de nœud
*	Sélectionne tous les éléments enfants du contexte de nœud.
@name	Sélectionne le nom d'attribut du contexte de nœud
@*	Sélectionne tous les attributs du contexte de nœud
row[1]	Sélectionne la première ligne enfant du nœud de contexte
row[last()]	Sélectionne la dernière ligne enfant du nœud de

	contexte
<code>*/row</code>	Sélectionne toutes les lignes enfants des petits-enfants du nœud de contexte
<code>/grid/row[5]/name[1]</code>	Sélectionne le premier nom de la cinquième ligne de la grille
<code>row//name</code>	Sélectionne l'élément 'nom' des descendants de la ligne élément enfant du nœud de contexte
<code>//name</code>	Sélectionne tous les noms de descendants du document racine et de cette façon sélectionne tous les éléments 'nom' dans le même document comme un nœud de contexte
<code>//row/name</code>	Sélectionne tous les éléments nom dans le même document comme un nœud de contexte qui a une ligne parent
<code>.</code>	Sélectionne le nœud de contexte
<code>.//name</code>	Sélectionne l'élément 'nom' des descendants du nœud de contexte
<code>row[@name="Caveman"]</code>	Sélectionne toutes les lignes du nœud de contexte qui ont un attribut de nom avec la valeur Caveman
<code>row[name]</code>	Sélectionne les lignes enfants du contexte de nœud qui ont un attribut ou plus, du nom des enfants
<code>row[@name and @price]</code>	Sélectionne toutes les lignes enfants du contexte de nœud qui ont déjà ensemble un attribut de nom et un attribut de prix

Comme vous pouvez le voir dans le tableau ci-dessus, l'axe `child` est l'option par défaut et l'on peut l'exclure de la syntaxe.

Dans le but de compléter ce court sommaire de XPath, vous devez savoir quels sont les opérateurs disponibles et quels opérandes peuvent être utilisés, les fonctions sur les nœuds, les fonctions sur les nombres, et les fonctions sur les chaînes de caractères.

Tableau C.3 - Opérateurs utilisés dans XPath

Opérateur	Description	Résultat
<code>and, or</code>	Et, ou logique	Vrai ou faux
<code>=, !=, &gt;, &gt;=, &lt;, &lt;=</code>	Égal, non égal, plus grand que, plus grand que ou égal à, moins que, moins que ou égal à	Vrai ou faux
<code>+, -, *, div, mod</code>	Plus, moins, multiplier, diviser, reste de la division (mode)	Nombre
<code> </code>	Comparaison de deux nœuds	Un nouveau nœud

Pour les fonctions qui peuvent être utilisées sur les nœuds, vous pouvez vous référer au tableau suivant :

Tableau C.4 - XPath : les fonctions sur les nœuds

Fonction	Description
last()	Retourne un nombre égal à la longueur du contexte de l'évaluation de l'expression du contexte.
position()	Retourne un nombre égal à la position du contexte de l'évaluation de l'expression du contexte.
count(node-set)	Retourne le nombre de nœuds dans la ligne d'arguments.

Pour les fonctions sur les nombres, une courte liste des fonctions que l'on peut trouver ci-dessous :

Tableau – C.5 - XPath : les fonctions sur les nombres

Fonction	Description
sum(node-set)	Retourne la somme qui est le résultat de la conversion d'une chaîne de caractères du nœud en nombre, pour chaque nœud dans la ligne d'arguments.
number(object ?)	Convertit ses arguments en nombre ; si l'argument est omis, c'est par défaut un noeud avec un contexte de noeud comme seul membre.
ceiling(nombre)	Retourne le plus petit nombre (le plus proche de moins l'infini) qui n'est pas moins grand que l'argument et qui est un entier.
floor(nombre)	Retourne le plus grand nombre qui n'est pas plus grand que l'argument et qui est un entier.

Pour les fonctions sur les chaînes de caractères nous pouvons utiliser quelques-unes de celles qui sont listées ci-dessous :

Tableau C.6 - XPath les fonctions sur les chaînes de caractères

Fonction	Description
string(object ?)	Convertit un objet en chaîne de caractères ; si l'argument est omis, c'est par défaut un ensemble de nœud avec un nœud de contexte comme seul membre.
concat(string, string, string*)	Retourne une concaténation de ces arguments
starts-with(string, string)	Retourne vrai, si le premier argument de la chaîne de caractères commence avec la chaîne de caractère du second argument, autrement retourne faux.
string-length(string ?)	Retourne le nombre de caractères de la chaîne de caractères ; si l'argument est omis, par défaut le noeud de contexte est converti en chaîne de caractères, en d'autres mots : un contexte de nœud en chaîne de caractères.
substring(string, number, number ?)	Retourne la sous-chaîne de caractères du premier argument démarrant à une position indiquée dans le second argument pour une longueur indiquée dans le troisième argument.

Contains(string, string)

Retourne vrai, si le premier argument contient la chaîne de caractères du second, et autrement retourne faux.

Pour en savoir plus sur XPath référez-vous à [www.w3.org/TR/xpath](http://www.w3.org/TR/xpath) or [www.w3schools.com](http://www.w3schools.com).

## XSLT

XSLT est utilisé pour la transformation des documents XML. Il peut transformer un document XML en un autre document XML, ou en un document HTML, ou même en un fichier texte. Après avoir vu de quoi il est capable vous y réfléchirez à deux fois lorsque vous serez face au problème lié à la conversion d'un fichier XML ou en l'utilisant comme source de données. Ce n'est pas aussi simple, mais si vous y faites attention, vous serez capable d'avoir dans vos mains une super arme pour le traitement des documents XML.

XSLT, comme sous-langage de XSL, est un langage basé sur XML, ainsi nous travaillons avec la bonne vieille structure XML. Dans le but de comprendre ce sur quoi nous travaillons, nous avons besoin d'avoir les idées claires sur la signification du mot *transformation*. Ainsi, la transformation fait référence aux règles de transformation d'une source d'un arbre XML en un arbre de résultat. Dans le but de terminer cette transformation, nous utilisons un modèle basé sur un document de base. Chaque modèle est comparé avec certain élément de l'arborescence source et le modèle utilise ces éléments pour composer l'arborescence résultante. Une chose importante, à garder en mémoire, est que l'arborescence source et l'arborescence résultante peuvent différer significativement en terme de structure ; la transformation appliquée à la source peut réordonner, filtrer ou ajouter de nouveaux éléments de structure.

La transformation formulée dans XSLT est appelée une feuille de style. Cela vient naturellement parce qu'on définit la manière dont le document sera affiché ; d'où son nom.

Parce que chaque document XSLT est un document XML bien formé, cela commence par l'habituelle ligne :

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

Après l'avoir déclaré comme un document XML, nous avons besoin d'aller plus loin et de spécifier que nous sommes en train de travailler avec un document de type feuille de style. L'élément racine peut être un des deux éléments : `<xsl:stylesheet>` ou `<xsl:transform>`, qui indique à l'analyseur quelle version de XSLT nous avons l'intention d'utiliser et quel espace de nom nous utilisons pour notre XSL :

```
| <xsl:stylesheet version="1.0"
| xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
|     et :
```

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Notez que nous utiliserons la version 1.0 de la recommandation XSLT et nous définissons un espace de nom `xsl` que nous utiliserons pour le reste du document. L'espace de nom `xsl` sera le préfixe pour nos expressions XSL. L'espace de nom nous permet d'accéder à ces attributs, éléments, et caractéristiques.

Un document de type feuille de style contient un ensemble de modèles de règles. Chacun de ces modèles de règles est fait en deux parties : un modèle qui est comparé avec les nœuds de l'arborescence source, et un modèle qui peut être appelé pour former une partie de l'arborescence résultante. Quand le compilateur XSLT reçoit un document XML qui doit être traité suivant le document de type feuille de style, il regarde chaque nœud du document XML et le vérifie vis-à-vis du modèle de règles qui peut former une partie de l'arborescence résultante. S'il y a une correspondance, le compilateur renvoie le modèle de la règle.

L'élément correspondant au modèle de la règle est `xsl:template`. Le modèle est spécifié dans l'attribut `match` de l'élément `xsl:template`. Le modèle du contenu affiché est spécifié à l'intérieur de l'élément `xsl:template`. Les instructions qui réalisent la transformation effective de l'arborescence source vers l'arborescence résultante sont les éléments de XSLT. Tous les éléments qui n'incluent pas le préfixe `xsl:` sont une partie de l'arborescence résultante.

```
<xsl:template match="expression">
```

C'est l'une des bases des éléments XSLT pour le document de type feuille de style. Dans le but de faire correspondre l'intégralité du document, nous utiliserons `match="/"`. Un autre attribut important est `name` qui peut être utilisé en donnant un nom à notre modèle dans le cas où nous avons besoin de l'appeler comme nous le verrons plus tard dans le chapitre.

Un autre élément, qui est très utile, est :

```
<xsl:for-each select="expression">
```

Cet élément passe à travers tous les nœuds retournés par l'expression XPath contenue dans l'attribut `select`. Il peut être facilement comparé à une instruction `for` dans PHP, où l'attribut sélectionné peut être la condition pour lequel le corps du `for` est exécuté. En mettant `select='data/grid/row'` nous sélectionnons toute les lignes de notre XML qui contient le produit.

```
<xsl:value-of select="expression"/>
```

L'élément `xsl:value-of` extrait la valeur du noeud sélectionné et la copie pour l'affichage. Par exemple :

```
<xsl:for-each select="data/grid/row">
  <xsl:value-of select="name"/>
</xsl:for-each>
```

On afficherait les noms de tous les produits de notre fichier source XML :

| `xsl:attribute`

Le `xsl:attribute` est utilisé pour la définition de l'attribut d'un élément. Par exemple :

| `<img><xsl:attribute name="src"><xsl:value-of  
select="src"/></xsl:attribute></img>`

On ajouterait l'attribut `src` à un élément HTML `img` et on définirait la valeur `src` à la valeur du produit courant. Pour que cet exemple fonctionne, nous aurions défini chaque produit dans notre fichier XML initial comme ceci :

| `<row>  
  <product_id>1</product_id>  
  <name>Santa Costume </name>  
  <price>14.99</price>  
  <on_promotion>1</on_promotion>  
  <src>picture.jpg</src>  
</row>`

Si nous avons parlé d'ajouter un attribut à un élément, nous avons besoin aussi de savoir que nous pouvons définir notre propre élément :

| `<xsl:élément name="element-name">`

L'`xsl:élément` peut être utile si nous devons définir notre propre noeud dans l'arbre résultant. L'attribut `nom` et ce nom seront affichés comme une vue.

| `<xsl:if test="expression">`

L'élément `xsl:if` est utilisé pour tester une condition par rapport aux noeuds de l'arborescence source et affiche quelque chose si la condition de test est évaluée à vrai.

| `<xsl:if test=" on_promotion &gt ; 0">  
  <input type="checkbox" name="on_promotion" disabled="disabled"  
  checked="checked" />  
</xsl:if>`

L'exemple donné affiche juste une boîte à cocher validée seulement si le produit est en promotion. Dans le but d'avoir quelque chose, si la condition du test est évaluée à faux, nous avons besoin d'utiliser un autre élément `xsl:if` ou un élément `xsl:choose`.

| `xsl:choose / xsl:when / xsl:otherwise`

Parfois, l'analogie fonctionne mieux, `xsl:choose` / `xsl:when` / `xsl:otherwise` implémente la même fonctionnalité comme `switch/case/default` en PHP ou C++. Voici un exemple :

```
<xsl:choose>
  <xsl:when test="on_promotion &gt ; 0">
    <input type="checkbox" name="on_promotion" disabled="disabled"
checked="checked"/>
  </xsl:when>
  <xsl:otherwise>
    <input type="checkbox" name="on_promotion" disabled="disabled" />
  </xsl:otherwise>
</xsl:choose>
```

L'exemple ci-dessus implémente la même fonctionnalité qu'une déclaration `if`, c'est un simple exemple, nous pourrions imaginer de prendre `xsl:choose`, qui est une alternative pour de multiples éléments `xsl:if`. Cela affiche une boîte à cocher validée si le produit est en promotion ou dévalidée dans le cas contraire.

```
<xsl:call-template name="template-name">
```

L'élément `xsl:call-template` appelle un modèle nommé :

```
<xsl:template match="/data/grid/row" name="row">
...
</xsl:template>
<xsl:call-template name="row"/>
```

L'exemple au dessus appelle un modèle nommé `row` pour toutes les lignes de notre fichier XML de départ.

Plus d'informations sur XSLT peuvent être trouvées à [www.w3.org/TR/xslt](http://www.w3.org/TR/xslt), [www.w3schools.com](http://www.w3schools.com), ou [www.topxml.com](http://www.topxml.com).

## TEST DE XPATH ET XSLT

Parce que les navigateurs modernes incluent le support de XPath et de XSLT, vous pouvez utiliser votre navigateur pour tester vos connaissances sur XSL.

Pour commencer, vous avez besoin de créer un fichier XML qui référence le fichier XSL. Créez un nouveau fichier nommé `products.xml`. (Son code est le même que le code dans la section *Setup* de l'annexe à l'exception des lignes accentuées.) Ajoutez le code suivant :

```
< ?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="products.xsl"?>
<data>
  <params>
    <returned_page>1</returned_page>
    <total_pages>6</total_pages>
```

```

<items_count>56</items_count>
<previous_page></previous_page>
<next_page>2</next_page>
</params>
<grid>
  <row>
    <product_id>1</product_id>
    <name>Santa Costume </name>
    <price>14.99</price>
    <on_promotion>1</on_promotion>
  </row>
  <row>
    <product_id>2</product_id>
    <name>Medieval Lady</name>
    <price>49.99</price>
    <on_promotion>1</on_promotion>
  </row>
  <row>
    <product_id>3</product_id>
    <name>Caveman</name>
    <price>12.99</price>
    <on_promotion>0</on_promotion>
  </row>
  <row>
    <product_id>4</product_id>
    <name>Costume Ghoul</name>
    <price>18.99</price>
    <on_promotion>0</on_promotion> </row>
  <row>
    <product_id>5</product_id>
    <name>Ninja</name>
    <price>15.99</price>
    <on_promotion>0</on_promotion>
  </row>
  <row>
    <product_id>6</product_id>
    <name>Monk</name>
    <price>13.99</price>
    <on_promotion>0</on_promotion>
  </row>
  <row>
    <product_id>7</product_id>
    <name>Elvis Black Costume</name>
    <price>35.99</price>
    <on_promotion>0</on_promotion>
  </row>
  <row>
    <product_id>8</product_id>
    <name>Robin Hood</name>
    <price>18.99</price>
    <on_promotion>0</on_promotion>
  </row>
  <row>
    <product_id>9</product_id>
    <name>Pierot Clown</name>

```

```

    <price>22.99</price>
    <on_promotion>1</on_promotion>
</row>
<row>
  <product_id>10</product_id>
  <name>Austin Powers</name>
  <price>49.99</price>
  <on_promotion>0</on_promotion>
</row>
</grid>
</data>

```

Créez aussi dans le même répertoire, un fichier nommé products.xsl, avec ce code :

```

< ?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
    <body>
    <h2>AJAX Grid</h2>
    <table style="border:1px solid black ;">
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Price</th>
        <th>Promo</th>
      </tr>
      <xsl:for-each select="data/grid/row">
        <xsl:élément name="tr">
          <xsl:attribute name="id">
            <xsl:value-of select="product_id" />
          </xsl:attribute>
          <td><xsl:value-of select="product_id" /></td>
          <td><xsl:value-of select="name" /> </td>
          <td><xsl:value-of select="price" /></td>
          <td>
            <xsl:choose>
              <xsl:when test="on_promotion > 0">
                <input type="checkbox" name="on_promotion"
disabled="disabled" checked="checked"/>
              </xsl:when>
              <xsl:otherwise>
                <input type="checkbox" name="on_promotion"
disabled="disabled"/>
              </xsl:otherwise>
            </xsl:choose>
          </td>
          <td>
            <xsl:élément name="a">
              <xsl:attribute name = "href">#</xsl:attribute>
              Edit
            </xsl:élément>
          </td>
        </tr>
      </xsl:for-each>
    </body>
  </template>

```

```

        </td>
      </xsl:élément>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Le chargement de products.xml dans le navigateur web comme Internet Explorer ou FireFox vous montrera la liste des produits formatée avec XSLT, comme le montre la figure C.1 :

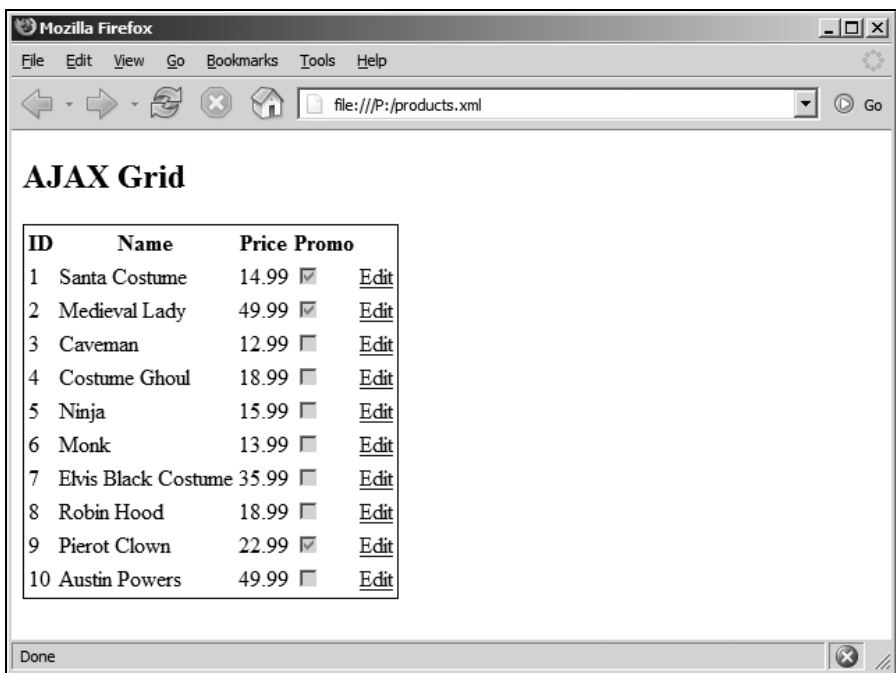


Figure C.1 - XML Transformé