6

Les comportements serveur XSLT

L'utilisation du langage XML est maintenant incontournable pour tous les créateurs d'applications multimédias. En effet, sa simplicité, sa souplesse et surtout son interopérabilité en font un langage universel permettant à la fois de structurer et d'échanger des informations, quel que soit le type de machine ou d'application. Tout cela, Macromedia l'a bien compris et a décidé d'ajouter, à sa version 8 de Dreamweaver, un nouveau comportement de transformation XSLT qui facilitera grandement l'intégration des flux de données XML, dans vos futurs sites dynamiques.

Avant de détailler l'usage de ce comportement, nous vous proposons une introduction au XML, car la connaissance des fondements de ce langage est indispensable à la compréhension des mécanismes de transformation XSLT. Si vous êtes déjà initié au XML, vous pouvez évidemment passer directement à la partie qui traite des transformations XSLT.

Avertissement

Ce chapitre n'a pas la prétention d'être un cours complet sur les transformations XSLT, car il faudrait y consacrer un livre entier pour en faire le tour. Notre modeste objectif est de vous fournir les concepts de base sur l'usage des expressions XSLT, afin que vous puissiez comprendre des comportements serveur XSLT simples et que vous puissiez analyser, voire modifier, ceux de Dreamweaver (même si leur utilisation ne nécessite pas la connaissance du langage XSLT). Que les puristes m'excusent si certaines définitions sont simplifiées à l'extrême ou tronquées, pour allègemer au maximum les apports théoriques, au profit des exemples pratiques. Ce chapitre est donc une simple « entrée en matière » et j'invite les personnes désirant approfondir le sujet à se référer à des ouvrages dédiés à cette technologie.

Introduction au XML

XML signifie eXtensible Markup Language. Comme le HTML, le XML est une norme SGML (Standard Generalized Markup Language), mais qui a été développée bien plus tard (en 1998, alors que le HTML était déjà normé par le consortium W3C depuis 1990).

Même si l'on a tendance à le présenter comme le successeur du HTML, le XML se caractérise par le fait qu'il contient uniquement des données structurées, mais aucune indication quant à leur présentation. Ainsi, si vous ouvrez un document XML dans un navigateur, il n'affichera que la structure des données sous forme d'arborescence, contrairement au document HTML qui montrera la traditionnelle page Web, car il contient à la fois les données et toutes les indications pour leur mise en forme.

XML est donc particulièrement bien adapté pour structurer, enregistrer et transmettre des données.

Les avantages du XML

Les avantages du XML sont multiples. Cependant, nous avons décidé de n'en énumérer que quelques-uns afin de vous convaincre de son intérêt.

1. **Simple** : comme son cousin le HTML, le XML est un simple document texte construit à partir de balises qui contiennent des informations. Il est donc lisible et interprétable par tous sans outil spécifique et avec peu de connaissances préalables.
2. **Souple** : l'utilisateur peut, s'il le désire, structurer les données et nommer librement chaque balise et attribut du document (contrairement au HTML pour lequel les noms des balises et des attributs sont prédéfinis).
3. **Extensible** : le nombre de balises n'est pas limité (comme c'est le cas pour le HTML) et pourra donc être étendu à volonté.
4. **Indépendant** : grâce à son contenu basé sur un document texte et donc universel, il peut être utilisé sur tout type de plate-forme (PC, Mac, Unix...) mais également quel que soit le langage de programmation (PHP, ASP...).
5. **Interopérabilité** : le fait que le XML soit un langage universel permet de favoriser l'interopérabilité des applications, en permettant de réaliser rapidement et simplement des échanges de données.
6. **Gratuit** : le XML étant développé par le consortium W3C, son utilisation est donc libre et ne nécessite pas l'achat de licence commerciale.

Structure d'un document XML

Toute personne ayant déjà utilisé un document HTML ne sera pas dépaysée à la vue du contenu d'un document XML (voir l'exemple).

Pour illustrer la structure d'un document XML, vous trouverez ci-dessous un exemple qui permet de stocker, d'une manière structurée, les coordonnées des différents propriétaires d'un immeuble de 2 étages, constitué de deux bâtiments A et B. La ventilation des propriétaires dans les étages de l'immeuble est illustrée par le dessin de la figure 6-1. Il est intéressant de comparer ce dessin avec la représentation XML de la figure 6-2, afin de mieux comprendre les mécanismes qui structurent les données d'un fichier XML.

Figure 6-1
Dessin représentant la ventilation des propriétaires dans les étages de l'immeuble.

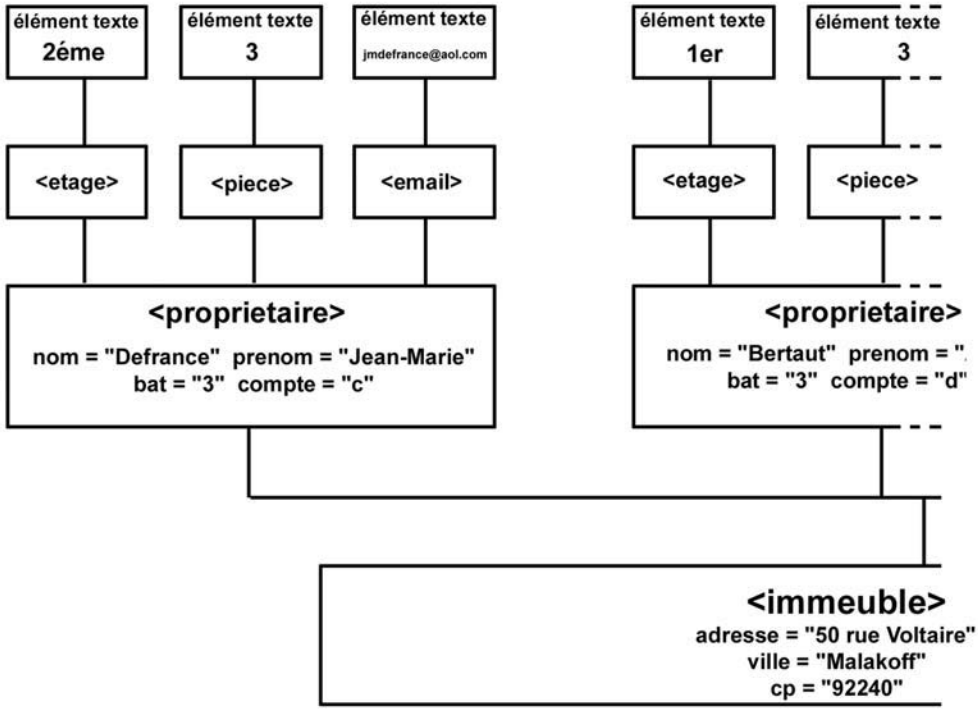

Figure 6-2
Structure du document XML correspondant à la représentation de l'immeuble.

Exemple de document XML :

```
<?xml version="1.0" encoding="iso-8859-1"?>
     <immeuble  adresse="50 rue Voltaire" ville="Malakoff" cp="92240">
       <proprietaire prenom="Jean-Marie" nom="Defrance" bat="a" compte="c">
           <etage>2eme</etage>
           <piece>3</piece>
           <email>jmdefrance@aol.com</email>
       </proprietaire>
       <proprietaire prenom="Alain" nom="Bertaut" bat="a" compte="d">
           <etage>1er</etage>
           <piece>3</piece>
           <email>abertaut@aol.com</email>
       </proprietaire>
       <proprietaire prenom="David" nom="Tardiveau" bat="a" compte="c">
```

```
            <etage>Rdc</etage>
            <piece>2</piece>
            <email>dtardiveau@aol.com</email>
        </proprietaire>
        <proprietaire prenom="Jean-Pierre" nom="Tavan" bat="b" compte="c">
            <etage>2eme</etage>
            <piece>2</piece>
            <email>jptavan@aol.com</email>
        </proprietaire>
        <proprietaire prenom="Bertrand" nom="Autin" bat="b" compte="d">
            <etage>1er</etage>
            <piece>2</piece>
            <email>bautin@aol.com</email>
        </proprietaire>
        <proprietaire prenom="Mario" nom="Fionda" bat="b" compte="d">
            <etage>Rdc</etage>
            <piece>1</piece>
            <email>mfionda@aol.com</email>
        </proprietaire>
</immeuble>
```

L'en-tête

Le document commence par un prologue (non obligatoire) contenant des informations sur la version de XML (**version="1.0"**), le jeu de caractères utilisés (**encoding="**iso-8859-1**"**) et l'autonomie d'un document (**standalone="no"**). Dans le prologue, seule la version est obligatoire. Si aucun type de codage n'est défini, l'UTF-8 est pris par défaut.

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

L'en-tête peut aussi faire référence à une déclaration du type de document (la DTD : Document Type Définition) qui permet d'en valider la conformité en se référant à l'URL d'un document en ligne ou en local (exemple : **http://adressedusite.com/info.dtd**).

```
<!DOCTYPE  info SYSTEM "http://adressedusite.com/info.dtd">
```

Si l'en-tête se réfère à une DTD externe – comme c'est le cas dans l'exemple ci-dessus –, le document n'est pas autonome et l'attribut **standalone** doit être configuré avec la valeur **"no"**. Dans le cas contraire (s'il n'y a pas de DTD ou si elle est interne), le document est autonome et la valeur de l'attribut **standalone** doit être définie à **"yes"**. En cas d'absence de l'attribut **standalone**, la valeur par défaut est **"no"**.

Le document XML qui suit l'en-tête utilise des blocs de construction semblables à ceux des documents HTML pour structurer son contenu. On retrouvera ainsi des éléments, des attributs, des valeurs et des commentaires.

L'élément

Un *élément* (appelé aussi *nœud*) est l'entité de base d'un document XML. Il peut contenir à son tour d'autres éléments ou tout type de contenu (chaîne de caractères ou autres). Le contenu d'un élément est encadré par une balise ouvrante (exemple : `<proprietaire>`) et une balise fermante (celle-ci contient le même nom que celui de la balise ouvrante, mais précédé d'un slash ex : `</proprietaire>`).

Si l'élément ne possède pas de contenu, alors les balises ouvrante et fermante sont remplacées par une seule et unique balise ayant la particularité d'avoir un slash à la fin du nom de l'élément (exemple : `<email />`).

Le nom indiqué dans ces deux balises doit décrire le contenu de l'élément, mais il n'est pas prédéfini comme en HTML (`<body>`, `<table>`, `<form>`, etc.). S'il est libre, il doit cependant comprendre uniquement des lettres de l'alphabet, des chiffres ou les caractères « - » et « _ », mais il ne doit jamais contenir d'espace ou commencer par un chiffre.

L'attribut

Il est possible d'ajouter des attributs à la balise ouvrante d'un élément (exemple : `<proprietaire nom="Defrance" >`). Les noms des attributs contenus dans une balise sont couplés avec une valeur encadrée par des guillemets (exemple : `nom="Defrance"`). Un attribut doit toujours avoir une valeur. Le nombre d'attributs par élément n'est pas limité, à condition que chaque nom d'attribut soit différent (l'exemple ci-après est donc incorrect : `<proprietaire nom="Durand" nom="Dupond">`). Si un élément a plusieurs attributs, ils doivent alors être séparés par des espaces (exemple : `<proprietaire prenom="Jean-Marie" nom="Defrance">`.

Les valeurs

Dans un document XML, les valeurs peuvent correspondre à des valeurs d'attribut (comme nous venons de le présenter précédemment : `nom="Defrance"`) ou à des valeurs d'élément (exemple : `<etage>2eme</etage>`). Attention, il est important de noter que la valeur d'un élément doit être considérée comme un composant texte enfant à part dans la hiérarchie du document XML (voir le bloc élément texte représentant ce `nodeValue` dans la figure 6-2).

Les commentaires

Comme pour le HTML, des commentaires peuvent être ajoutés dans un document XML. La syntaxe est d'ailleurs identique à celle utilisée pour en intégrer dans une page HTML (exemple : `<!--Ceci est un commentaire XML-->`). À l'intérieur d'un commentaire, vous pouvez utiliser tout type de symbole, sauf les doubles tirets « -- » . Les commentaires servent évidemment à annoter les documents XML pour se souvenir de l'utilité de certains blocs

d'éléments ou pour détailler la structure du document, mais ils peuvent servir aussi à déboguer en neutralisant une partie du document afin qu'il ne soit pas visible par l'analyseur XML.

Règles d'écriture d'un document XML bien formé

Même si les documents XML sont simples et extensibles, ils n'en sont pas pour autant démunis de règles. On appelle « document bien formé » les documents qui les respectent. La partie suivante énumère les principales conventions qu'il est indispensable de respecter dans vos futurs développements.

Une méthode simple pour savoir si un document est bien formé est de l'appeler avec un navigateur Internet récent, c'est-à-dire possédant un interpréteur XML intégré tel que les navigateurs de versions ultérieures à IE 5 ou à Netscape 6 (voir figure 6-2).

Un seul élément racine : chaque document XML ne doit posséder qu'un seul élément racine. L'élément racine est caractérisé par le fait qu'il contient tous les autres éléments du document. Ce composant particulier, s'appelle « nœud racine » ou « root ».

Exemple : `<immeuble><proprietaire>Fionda</ proprietaire >< proprietaire >Tardiveau</ proprietaire ><immeuble>` (ici la balise `immeuble` est le nœud racine du document XML).

Des balises de fermeture obligatoires : nous avons vu précédemment que chaque élément doit être encadré par des balises ouvrantes et fermantes. Contrairement au HTML (par exemple la balise `<p>` n'est pas obligatoirement fermée en HTML, de même que `<hr>` est une balise inhérente sans balise de fermeture), le XML ne supporte pas l'absence des balises fermantes. Il conviendra donc de veiller à toujours ajouter une balise de fermeture à tous les éléments d'un document XML. Si le document possède un élément vide, il faudra utiliser une balise unique, mais avec un slash avant le signe `>` final (exemple : `<email />`).

Respecter l'imbrication des éléments : lorsque vous ouvrez un premier élément puis un second, il faut veiller à clore la balise de fermeture du second avant celle du premier. Ainsi le code suivant est incorrect : `<a><b>contenu</a></b>` alors que celui-ci est correct `<a><b>contenu</b></a>`.

Respecter la casse : le XML est sensible à la casse. Ainsi, les noms d'éléments « etage », « Etage » et « ETAGE » seront considérés comme différents en XML. Les noms des éléments et des attributs doivent donc être en minuscules.

Mettre les valeurs des attributs entre guillemets : si une balise contient un couple nom d'attribut et sa valeur, il conviendra de mettre toujours la valeur entre guillemets (simples ou doubles). Exemple : `<proprietaire nom="Defrance">`.

Utilisez les entités prédéfinies pour les caractères réservés : comme en HTML, il existe des caractères réservés dont l'usage est interdit (`<`, `>`, `&`, `'` et `"`). Pour chacun de ces caractères, il convient d'utiliser l'entité prédéfinie correspondante (`&lt;` , `&gt;` , `&amp;` , `&quot;` , `&apos;`).

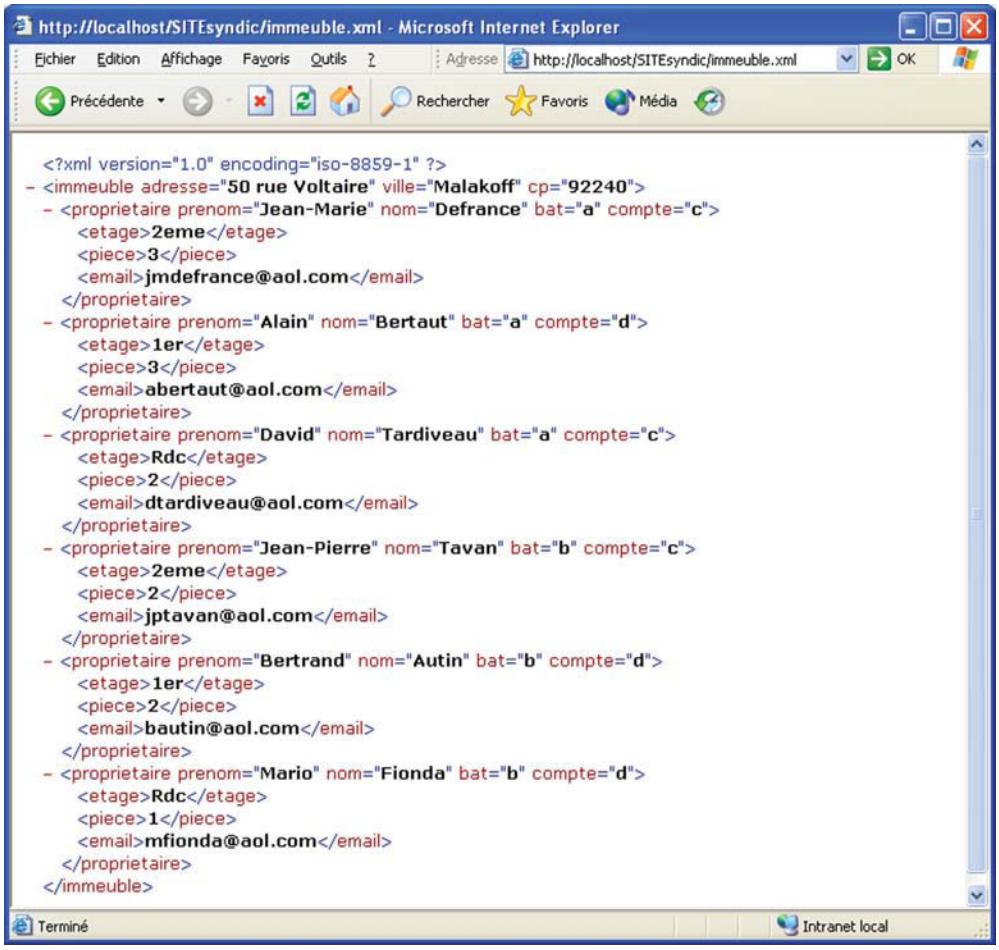

Figure 6-3
Affichage d'un document XML bien formé dans un navigateur Internet.

Utilisez une section CDATA pour contourner un bloc de texte complet : Afin d'éviter d'utiliser des entités pour des longs blocs de texte comportant des caractères réservés, vous pouvez ajouter une section CDATA en respectant la syntaxe suivante : `<![CDATA[bloc de texte]]>`

Les feuilles de style XSL

Les feuilles de style XSL (eXtended Style sheet Language) permettent de gérer la mise en page d'un document XML, et sont sur ce point semblables aux feuilles de styles CSS (Casca-

ding Style Sheets) qui permettent d'appliquer une mise en forme à un document HTML. Cependant, les feuilles de style XSL peuvent aussi être exploitées côté serveur et intégrer des fonctionnalités beaucoup plus puissantes que les CSS qui permettent, par exemple, de transformer des données au format XML ou de gérer la pagination.

Sans détailler toutes les caractéristiques techniques des feuilles XSL, il est cependant important de savoir que XSL est surtout une famille de spécifications comprenant trois sous-ensembles :

1. XSLT (XSL Transformation) : qui permet, comme son nom l'indique, la transformation d'un document XML.
2. X-Path : qui permet de naviguer à l'intérieur d'un document XML pour localiser certains objets afin de leur appliquer un traitement.
3. XSL-FO (XSL Formating Objects) : qui permet de décrire la présentation d'un document.

En effet, pour publier tout ou partie d'un document XML source, il faut commencer par localiser l'information (c'est le rôle de X-Path), puis la transformer dans le format d'un langage de présentation (c'est le rôle de XSLT), et enfin construire le langage de présentation final en incluant l'information sélectionnée (cela peut être le rôle de XSL-FO, notamment dans le cas d'un format final PDF, mais en général on se contente d'exploiter le XHTML pour un affichage dans un navigateur).

Dans le cadre de cet ouvrage, nous n'aborderons pas le langage XSL-FO, mais nous exploiterons de nombreuses feuilles XSLT qui intégreront des instructions X-Path. Nous commencerons donc, dans la partie suivante, par définir quelques notions de base sur le langage X-Path, afin de pouvoir les appliquer dans des exemples pratiques de transformations XSLT à la fin de ce chapitre.

Le langage de navigation X-Path

X-Path, comme son nom l'indique, est un langage permettant de définir des « chemins d'accès », pour désigner des objets cibles dans un document XML. Ces objets ainsi localisés (pouvant être tout ou partie du document XML) pourront ensuite être traités par le fichier XSLT, avant d'être renvoyés à l'application cliente. Les expressions X-Path sont donc souvent intégrées directement dans le fichier XSLT, à tel point qu'on a souvent tendance à les confondre. Mais sachez que X-Path est un langage à part entière pouvant d'ailleurs être utilisé dans d'autres contextes. En pratique, les expressions X-Path employées dans les fichiers XSLT sont généralement assez simples. Nous n'étudierons donc pas tout le langage X-Path d'une manière exhaustive, mais nous nous limiterons à ses principales expressions qui permettront déjà de traiter la grande majorité des projets de transformation.

Mode de lecture

Les expressions X-Path seront par la suite utilisées dans les fichiers XSLT générés par le comportement serveur XSLT de Dreamweaver. Il n'est pas indispensable de connaître la syntaxe du langage X-Path pour pouvoir créer vos premiers comportements. Si cette partie dédiée à la présentation de X-Path vous semble trop ardue ou trop théorique, vous pouvez passer directement à la pratique dans la partie consacrée à la création des comportements serveur XSLT, et revenir ensuite à ce passage, en deuxième lecture, pour analyser le code généré par Dreamweaver ou l'adapter à vos besoins.

Modèle arborescent utilisé par X-Path

Lorsque X-Path parcourt un document XML, il modélise sa structure arborescente sous la forme d'un arbre, selon sept types de nœuds différents. Cet arbre permet ensuite à X-Path d'identifier des objets du document XML selon leur type grâce à différentes expressions spécifiques.

Voici ci-dessous les sept types de nœuds, et quelques exemples d'expressions X-Path permettant de reconnaître un objet du document XML selon son type :

Document : (ou `root` souvent représenté par une barre oblique : `/`) est le type du nœud racine de l'arbre XML. Attention, `root` n'est pas l'élément racine visible du document XML mais son père, et il n'est pas visible dans le document.

Element : est le type d'un nœud élément. Un nœud élément peut être identifié par son nom, mais on peut aussi retrouver tous les éléments par rapport au nœud contexte à l'aide du caractère joker : `*`.

Text : est le type d'un nœud texte appartenant à un élément. Il est possible d'identifier tous les nœuds textuels par rapport au nœud contexte à l'aide de l'expression : `text()`.

Attribute : est le type d'un nœud attribut d'un élément. L'attribut peut être identifié par son nom précédé du symbole `@` comme `@nomAttribut` par exemple. À noter que l'on peut aussi retrouver tous les attributs d'un nœud contexte à l'aide de l'expression `@*`.

Namespace : est le type d'un nœud domaine. Le nœud domaine peut être identifié par son nom.

Processing-instruction : est le type d'un nœud processing-instruction qui correspond, en général, à une directive de traitement adressée à un processeur différent de XSLT.

Comment : est le type de nœud commentaire dans un document XML.

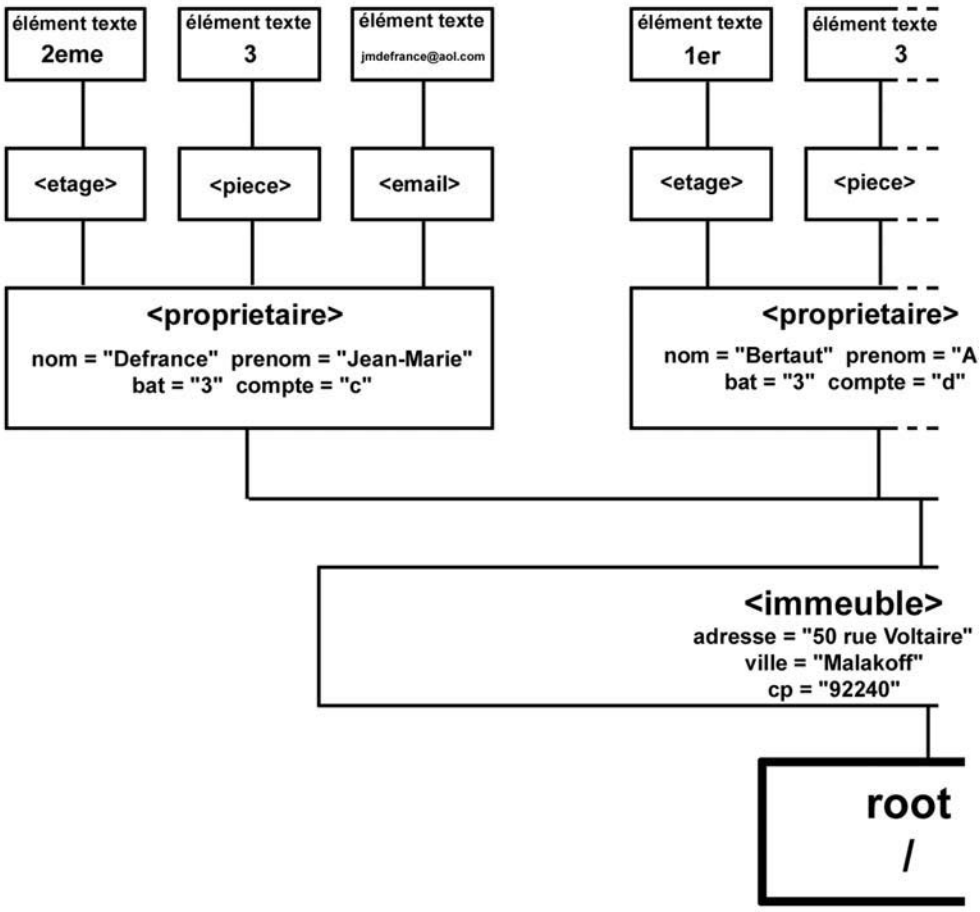

Figure 6-4
Arbre XML correspondant à l'exemple du fichier immeuble.xml présenté dans la partie précédente. Si vous comparez cette structure avec celle de la figure 6-2, vous remarquerez qu'elle diffère uniquement par la présence du nœud racine «root».

Syntaxe des expressions X-Path

Une expression X-Path permet de construire des chemins de localisation qui peuvent être constitués d'une seule étape ou d'une suite d'étapes. Dans le cas d'une suite d'étapes de localisation, chacune d'elles sera séparée par une barre oblique (/).

La présence d'une barre oblique initiale indique qu'il s'agit d'un chemin absolu prenant naissance à la racine de l'arbre. S'il n'y en a pas, il s'agit alors d'un chemin relatif débutant à l'endroit du nœud contexte.

Syntaxe d'un chemin de localisation absolu (référence à la racine de l'arbre) :

```
/étape1/étape2/.../étapeN
```

Syntaxe d'un chemin de localisation relatif au nœud contexte :

```
étape1/étape2/.../étapeN
```

Le nœud contexte

On utilise souvent, dans les expressions XSLT, le terme de « nœud contexte ». Cela signifie que l'expression X-Path sera évaluée d'une manière relative à ce nœud. Le nœud contexte est désigné, dans la syntaxe X-Path, par le point (.) ou encore par l'expression (version longue) `self::node()`.

Syntaxe d'une étape de localisation :

```
Axe  ::   Filtre   [prédicat1] [prédicat2] ...[prédicatN]
```

L'étape de localisation peut être décomposée en trois zones différentes :

1. **L'axe** (Node set) définit le sens du parcours, à partir du nœud contexte. Par défaut, l'axe est l'enfant direct du nœud contexte, soit `child`, mais il peut être aussi le parent du nœud contexte, `parent`, ses attributs, `attribute`, ses frères, `following-sibling` ou `preceding-sibling` (selon le sens de lecture du document), ou encore toute sa descendance, `descendant`, etc.
2. **Le filtre** (Node test), appelé aussi « déterminant », est une fonction booléenne qui détermine les nœuds qui seront retenus dans l'expression d'évaluation finale. Il existe plusieurs familles de filtres :
   - Si le filtre est un simple nom d'élément ou d'attribut, alors seuls les nœuds (éléments ou attributs) de l'axe choisi dont le nom est égal à celui du filtre seront retenus.
   - Si le filtre est `*` alors tous les nœuds de l'axe choisi seront retenus.
   - Enfin, le filtre peut sélectionner des nœuds de l'axe choisi selon leur type. Par exemple si le filtre est `text()` seront choisis les nœuds de type `text`, si le filtre est `comment()` seront retenus les nœuds de type `comment`, etc. Il existe cependant un cas particulier lorsque le filtre est `node()`. En effet, `node()` ne rejetant aucun nœud, dans ce cas sera sélectionné tout le contenu de l'axe choisi, sans restriction.
3. **Les prédicats** permettent d'affiner la sélection déjà opérée par les filtres en éliminant des nœuds choisis, ceux qui ne répondent pas aux critères exprimés par leur expression (les prédicats sont encadrés par des crochets `[` et `]`). Dans l'exemple suivant : `child::proprietaire[position( )=2]`, seul le second élément enfant dont le nom est

`proprietaire` sera sélectionné. S'il y a plusieurs prédicats, ils sont exécutés en cascade : le résultat d'un prédicat est à son tour soumis au prédicat suivant (de la gauche vers la droite), et ainsi de suite. Les chemins de localisation pouvant être rapidement très longs, il existe des abréviations que vous pourrez utiliser pour des constructions fréquentes (voir tableau 6-1).

Tableau 6-1 – Abréviations X-Path

Expression X-Path (version longue)	Correspondance (version courte)
child::*proprietaire*	proprietaire
child::*	*
attribute::*prenom*	@*prenom*
attribute::*	@*
[position( ) = n]	[n]
self::node( )	.
parent::node( )	..
/descendant-or-self::node( ) /	//

Union d'expressions

Dans certains cas, il est possible d'utiliser l'opérateur « | » pour unir deux expressions.

L'exemple ci-dessous sélectionne ainsi l'attribut « `prenom` » OU « `nom` » de l'élément contexte :

```
attribute::prenom | attribute::nom
```

Pour illustrer cette partie, nous proposons ci-dessous quelques exemples de chemins de localisation simples (voir tableau 6-2).

Tableau 6-2 – Exemples des chemins de localisation simples

Chemin (court)	Chemin (long)	Objet(s) sélectionné(s)
/	/	La racine du document.
proprietaire	child::*proprietaire*	Tous les éléments <*proprietaire*> qui sont les enfants directs du nœud contexte.
*	child::*	Tous les nœuds de type « Element » qui sont les enfants directs du nœud contexte.
@*prenom*	attribute::*prenom*	L'attribut *prenom* de l'élément contexte.
@*		Tous les attributs de l'élément contexte.
.		Le nœud contexte.

Tableau 6-2 – Exemples des chemins de localisation simples (*suite*)

Chemin (court)	Chemin (long)	Objet(s) sélectionné(s)
text()	child::text()	Tous les nœuds de type *Text* qui sont les enfants directs du nœud contexte.
proprietaire/@*prenom*	child::*proprietaire*/attribute::*prenom*	L'attribut *prenom* de l'élément <*proprietaire*>.
proprietaire [@*prenom*='jean']	child::*proprietaire*[attribute::*prenom*='jean']	Les éléments <*proprietaire*> ayant un attribut *prenom* égal à *jean* et qui sont les enfants directs du nœud contexte.
//*etage*	/descendant-or-self::node( )/child::*etage*	Tous les éléments <*etage*> avec racine comme ancêtre (donc tous les éléments <*etage*> du document XML).
../*proprietaire*	parent::node()/child::*proprietaire*	Tous les éléments <proprietaire> enfants d'un nœud quelconque parent du nœud contexte.

Le langage de transformation XSLT

Le XSLT est un langage permettant de transformer des documents XML vers des documents XHTML (cas le plus fréquent), mais aussi vers d'autres pages XML ou encore vers des types de formats adaptés aux différents canaux de visualisation (téléphone, WebTV...). XSLT est souvent utilisé conjointement avec X-Path, le langage de navigation du XSL.

Mode de lecture

Les instructions XSLT seront par la suite utilisées dans des fichiers XSLT générés par le comportement serveur XSLT de Dreamweaver. Il n'est pas indispensable de connaître la syntaxe des instructions XSLT pour pouvoir créer vos premiers comportements. Si cette partie dédiée à la présentation de la syntaxe XSLT vous semble trop ardue ou trop théorique, vous pouvez passer directement à la pratique dans la partie consacrée à la création des comportements serveur XSLT, et revenir ensuite à ce passage, en deuxième lecture, pour analyser le code généré par Dreamweaver ou le modifier pour l'adapter à vos besoins.

Processeur XSLT

XSLT est un langage interprété, et il faut donc disposer d'un interpréteur pour exploiter les fichiers XSLT. On appelle « Processeur XSLT » le mécanisme qui permet d'interpréter les fichiers XSLT. En pratique, on le trouve sur les serveurs d'application (comme par exemple avec PHP et son extension XSL) ou sur les navigateurs récents (comme Internet Explorer 6 ou Firefox 1.0.2 ou version ultérieure).

En résumé, pour qu'une transformation XSLT puisse être effectuée, il faut disposer du document source XML, du document XSLT et d'un processeur XSLT qui exécutera la transforma-