

XML pour les bibliothécaires : un manuel et un atelier

by Eric Lease Morgan

XML pour les bibliothécaires : un manuel et un atelier

by Eric Lease Morgan

Ce manuel est libre; vous pouvez le redistribuer et/ou le modifier selon les termes de la Licence Publique Générale GNU telle que publiée par la Free Software Foundation; soit dans sa version 2, soit, à votre convenance, toute version ultérieure.

Ce manuel est distribué avec l'espoir qu'il soit utile, mais SANS AUCUNE GARANTIE; sans même la garantie implicite qu'il puisse être commercialement intéressant ou bon à un usage particulier. Voir la Licence Publique Générale GNU pour plus de détails.

Vous devriez avoir reçu une copie de la Licence Publique Générale GNU avec ce manuel. Si tel n'est pas le cas, voir : The Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Merci à David Cherry, ami de longue date, qui a fourni les dessins. Lori Bowen Ayre mérite des applaudissements pour son soutien éditorial. Ce manuel a été sponsorisé par Infopeople. Roy Tennant a aidé à relire ce document. Merci! --ELM

Traduction française de Nicolas Morin (02/08/2003).

Pour des informations éventuellement plus à jour, voir la page d'accueil -en anglais- de Getting Started With XML [<http://infomotions.com/musings/getting-started/>].

Ceci est la seconde version de ce document, datée du lundi 21 avril 2003. La première version publique était datée du samedi 22 février 2003.

Table of Contents

Preface	
I. XML : généralités	
1. Introduction	
Qu'est-ce que XML, et en quoi cela m'intéresse?	3
2. Une introduction douce au marquage XML	
La syntaxe XML	6
Les documents XML ont toujours un et un seul élément racine	6
Les noms des éléments sont sensibles à la casse	7
Les éléments doivent toujours être fermés	7
Les éléments doivent toujours être correctement emboîtés	7
Les attributs d'éléments doivent toujours être entre guillemets	8
Il y a seulement cinq entités définies par défaut (<, >, &, ", et ')	8
La sémantique d'XML	8
Exercice - Vérifier la syntaxe XML	9
3. Créer votre propre système de balises	
But et éléments constitutifs	10
Exercice - Créez votre propre système de balises XML	12
4. Gérer l'affichage d'XML avec les feuilles de style	
Introduction	14
display	15
margin	16
text-indent	16
text-align	16
list-style	16
font-family	17
font-size	17
font-style	17
font-weight	18
Rassembler tout ça	18
Tables	19
Exercice - Afficher du XML avec CSS	20
5. Transformer XML avec XSLT	
Introduction	22
Afficher du texte	22
Afficher des données tabulées	26
Manipuler des données XML	28
Utiliser XSLT pour créer d'autres types de fichiers textuels	30
6. Les Document Type Definitions (DTD)	
Définir des vocabulaires XML avec les DTDs	32
Noms et nombres d'éléments	33
PCDATA	34
Séquences	34
Rassembler tout ça	34
Exercice - Ecrire une DTD simple	36
II. Quelques vocabulaires XML spécifiques	
7. XHTML	
Introduction	40
Exercice - Ecrire un document XHTML	43
8. TEI	
Introduction	45
Quelques éléments	45
Exercice	48
9. EAD	

Introduction	49
Exemple	50
10. DocBook	
Introduction	53
Traiter DocBook avec XSLT	55
11. RDF	
Introduction	59
Exercice	61
12. Récolter des métadonnées avec OAI-PMH	
Qu'est-ce que l'Initiative pour les Archives Ouvertes?	62
Le problème	62
La solution	63
Verbes	63
Réponses -- le flux XML	64
Un exemple	67
Conclusion	67
III. Annexes	
A. Quelques lectures	
XML en général	70
Cascading Style Sheets - CSS	70
XSLT	70
DocBook	70
XHTML	71
RDF	71
EAD	71
TEI	71
OAI-PMH	71

Preface

Destiné aux bibliothécaires et personnels des bibliothèques, cet atelier est une introduction au XML (extensible markup language) au travers de nombreux exemples propres aux bibliothèques, des démonstrations et des exercices structurés. Vous pourrez ainsi évaluer l'intérêt d'XML pour rendre les données et les informations de votre bibliothèque plus accessibles (aux hommes et aux ordinateurs). Dans les exemples, vous verrez : comment ajouter de la valeur à un texte électronique, faciliter la recherche dans des archives, et implémenter des pages web respectant les standards. A la fin, vous aurez eu droit à une introduction complète à XML et vous serez capables de : 1) énumérer les six règles qui régissent la syntaxe des documents XML, 2) créer votre propre langage de marquage, 3) écrire des documents XML en utilisant un bloc-notes et les valider en utilisant un navigateur web, 4) appliquer une mise en page et des techniques typographiques à des documents XML en utilisant les feuilles de style en cascade, 5) créer des documents XML simples en utilisant un certain nombre de vocabulaires XML standard importants dans le monde des bibliothèques tels que XHTML, TEI, et EAD, et enfin, 6) expliquer pourquoi XML est important pour les bibliothèques.

Les principaux points de ce manuel :

- les usages d'XML dans les bibliothèques pour créer, stocker, et diffuser des textes électroniques, des aides à la recherche d'archives, et des pages web
- Six règles simples pour créer des documents XML valides
- combiner les usages des feuilles de style et de documents XML pour afficher des données et de l'information dans un navigateur web
- utiliser XHTML et voir dans quelle mesure cela peut contribuer à rendre votre site web plus facilement accessible : à toutes sortes de gens et de robots
- Comment on peut créer automatiquement des pages web grâce à XSLT et ainsi permettre à la bibliothèque de transformer des documents XML en documents dans d'autres formats
- Enrichir des textes électroniques en utilisant le système de marquage TEI pour permettre à la bibliothèque d'ajouter de la valeur à des documents numériques
- Produire des aides à la recherche d'archives en utilisant EAD pour permettre à la bibliothèque de partager ses collections avec d'autres institutions

Ce manuel est divisé en chapitres comme suit:

1. Qu'est-ce que XML, et en quoi cela m'intéresse?
2. Une introduction douce au marquage XML
3. Créer votre propre système de balises
4. Gérer l'affichage d'XML avec les feuilles de style
5. Transformer XML avec XSL
6. Valider XML avec les DTD
7. Introduction à quelques langages XML: XHTML, TEI, DocBook
8. Partager des métadonnées avec RDF et OAI

9. Quelques lectures

Part I. XML : généralités

Table of Contents

1. Introduction	
Qu'est-ce que XML, et en quoi cela m'intéresse?	3
2. Une introduction douce au marquage XML	
La syntaxe XML	6
Les documents XML ont toujours un et un seul élément racine	6
Les noms des éléments sont sensibles à la casse	7
Les éléments doivent toujours être fermés	7
Les éléments doivent toujours être correctement emboîtés	7
Les attributs d'éléments doivent toujours être entre guillemets	8
Il y a seulement cinq entités définies par défaut (<, >, &, ", et ')	8
La sémantique d'XML	8
Exercice - Vérifier la syntaxe XML	9
3. Créer votre propre système de balises	
But et éléments constitutifs	10
Exercice - Créez votre propre système de balises XML	12
4. Gérer l'affichage d'XML avec les feuilles de style	
Introduction	14
display	15
margin	16
text-indent	16
text-align	16
list-style	16
font-family	17
font-size	17
font-style	17
font-weight	18
Rassembler tout ça	18
Tables	19
Exercice - Afficher du XML avec CSS	20
5. Transformer XML avec XSLT	
Introduction	22
Afficher du texte	22
Afficher des données tabulées	26
Manipuler des données XML	28
Utiliser XSLT pour créer d'autres types de fichiers textuels	30
6. Les Document Type Definitions (DTD)	
Définir des vocabulaires XML avec les DTDs	32
Noms et nombres d'éléments	33
PCDATA	34
Séquences	34
Rassembler tout ça	34
Exercice - Ecrire une DTD simple	36

Chapter 1. Introduction



Qu'est-ce que XML, et en quoi cela m'intéresse?

En une phrase XML (eXtensible Markup Language - ou langage de balisage extensible) est un standard qui donne les moyens de partager des données et de l'information entre ordinateurs et entre logiciels de la façon la moins ambiguë possible. Une fois l'information transmise, c'est le programme informatique qui reçoit qui doit interpréter les données pour les rendre utilisables et transformer ainsi ces données en information. Parfois cette information sera restituée sous forme d'une page HTML. Parfois elle sera utilisée pour interroger et/ou mettre à jour une base de données. Conçu initialement comme un outil de publication pour le web, XML s'est montré utile aussi pour des choses qu'on n'a jamais eu l'intention d'afficher sous forme de pages web.

Pensez à XML comme à quelque chose qui représenterait des fichiers textes structurés avec des tabulations. Ces derniers sont facilement lisibles par l'homme. Ils sont faciles à importer dans des traitements de texte, des bases de données, des tableurs. Une fois importés, leur structure simple rend leur contenu facile à manipuler. Les textes tabulés sont même indépendants des systèmes d'exploitation (pour autant que vous surmontiez le problème des différences entre les retour-chariots des machines Windows, Mac ou Unix). Voyez l'exemple ci-dessous:

Amanda	10	chien	marron
Bernard	12	chien	bleu
Jack	3	chat	noir
Naia	1	chat	marron
Stop	5	cochon	marron
Puce	14	chat	argent

Les textes structurés par des tabulations posent deux problèmes. D'abord, la signification de chacune des valeurs délimitée n'est pas explicitée. Pour savoir ce qu'est censée représenter chaque valeur on doit nécessairement avoir (ou savoir) à l'avance une sorte de carte de situation des données. Ensuite, et c'est plus important encore, les fichiers délimités par des tabulations peuvent seulement représenter une structure de données simple, une structure de données analogue à un simple canevas de lignes et de colonnes. Autrement dit, les fichiers délimités par des tabulations sont exactement comme des fichiers de base de données aplatis. Il n'y a aucun moyen simple, standardisé, de représenter les données de façon hiérarchisée.

A l'image des textes tabulés, les fichiers XML sont faciles à lire par l'homme puisqu'ils ne contiennent que des caractères Unicode -- une version très étendue de la table d'encodage originale ASCII. Qui plus est, les fichiers

XML sont indépendants du système d'exploitation et des applications, ce qui présente aussi l'avantage d'éliminer les problèmes de retour-chariot.

A la différence des textes tabulés, les fichiers XML déclarent explicitement la signification de chacune des valeurs que contient le fichier. La part de non-explicite est très faible. La valeur de chaque élément est explicitement décrite. XML transforme les données en information. Le fichier tabulé de la figure 1.1 est simplement une liste organisée de mots et de chiffres. Aucun contexte n'est explicité et mots et chiffres représentent seulement des données. Par contre dans un fichier XML les mots et les chiffres se voient attribué une valeur et un contexte, et sont ainsi transformés de données en informations. Qui plus est, il est facile de créer des structures de données hiérarchiques en utilisant XML. La figure 1.2 illustre ces concepts. Sans qu'un examen très approfondi soit nécessaire, on voit immédiatement que ces données représente une liste d'animaux (pets), en l'occurrence six animaux, et que chaque animal (pet) a un nom (name), un âge (age), une espèce (type), et une couleur (color). Etait-ce aussi évident dans l'exemple précédent?

```
<pets>
  <pet>
    <name>Puce</name>
    <age>14</age>
    <type>chat</type>
    <color>argent</color>
  </pet>
  <pet>
    <name>Amanda</name>
    <age>10</age>
    <type>chien</type>
    <color>marron</color>
  </pet>
  <pet>
    <name>Jack</name>
    <age>3</age>
    <type>chat</type>
    <color>noir</color>
  </pet>
  <pet>
    <name>Bernard</name>
    <age>12</age>
    <type>chien</type>
    <color>bleu</color>
  </pet>
  <pet>
    <name>Naia</name>
    <age>1</age>
    <type>chat</type>
    <color>marron</color>
  </pet>
  <pet>
    <name>Stop</name>
    <age>5</age>
    <type>cochon</type>
    <color>marron</color>
  </pet>
</pets>
```

A mesure que les économies actuelles se consacrent de plus en plus aux services, les données et l'information deviennent de plus en plus importantes. Parallèlement, les bibliothèques se préoccupent moins des livres et plus des idées et des concepts contenus dans les livres. Dans ces deux domaines, nous avons besoin d'un moyen de faire circuler les données et l'information efficacement. Des données XML partagées entre ordinateurs ou entre logiciels via HTTP représentent une méthode (en évolution) pour améliorer ce partage : cette méthode est appelée, de façon générique, Services Web.

Par exemple, un marquage XML nommé RSS (Rich Site Summary) est utilisé de façon croissante pour syndiquer des listes d'URL représentant des nouvelles tirées de sites web d'information. RDF (Resource Description Frame-

work) est un marquage XML utilisé pour encapsuler des métadonnées concernant les contenus qu'on trouve à la fin des URL. TEI (Text Encoding Initiative) et TEILite sont tous les deux des marquages SGML et XML utilisés pour donner des valeurs explicites à des éléments trouvés dans des oeuvres littéraires. De même un autre marquage XML appelé DocBook est de plus en plus utilisé pour baliser des livres ou des articles d'informatique. L'Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) utilise XML pour récupérer des métadonnées trouvées sur des sites web.

En tant que professionnels de l'information, il nous appartient d'apprendre à exploiter les possibilités d'XML, car XML rend beaucoup plus aisé de communiquer de l'information de façon non-ambigue et indépendamment des plateformes dans un monde en réseau. N'est-ce pas précisément de cela qu'il s'agit quand on parle de bibliothéconomie et de sciences de l'information?

Chapter 2. Une introduction douce au marquage XML



La syntaxe XML

Les documents XML ont des structures syntaxiques et sémantiques. La syntaxe (pensez à orthographe et à ponctuation) est constituée de règles minimales dont (la liste n'est pas exhaustive):

1. Les documents XML ont toujours un et un seul élément racine
2. Les noms des éléments sont sensibles à la casse
3. Les éléments doivent toujours être fermés
4. Les éléments doivent toujours être correctement emboîtés
5. Les attributs d'éléments doivent toujours être entre guillemets
6. Il y a seulement cinq entités définies par défaut (<, >, &, ", et ')

Chacune de ces règles est décrite plus en détail ci-dessous.

Les documents XML ont toujours un et un seul élément racine

La structure d'un document XML est une arborescence comprenant un tronc et éventuellement de nombreuses branches. Le tronc, unique, représente l'élément racine du document XML. Voyez le document HTML suivant (Figure 2.1), très simplifié:

```
<html>
<head>
  <title>Bonjour, tout le monde</title>
```

```
</head>
<body>
  <p>Bonjour, tout le monde</p>
</body>
</html>
```

La structure de ce document devrait vous être familière. C'est un document XML valide, et il contient un seul élément racine, en l'occurrence html. Il y a deux branches à ce document, head et body.

Les noms des éléments sont sensibles à la casse

Les noms des éléments, le vocabulaire élémentaire des documents XML, sont sensibles à la casse. La figure 2.1 comporte cinq éléments: html, head, title, body, et p. Puisque les noms des éléments sont sensibles à la casse, l'élément html n'est pas égal à l'élément HTML, ni à HTml ou Html. Ceci s'applique aux autres éléments.

Les éléments doivent toujours être fermés

Chaque élément est dénoté par des parenthèses ouvrante et fermante, les signes plus petit que (<) et plus grand que (>), respectivement. Les éléments XML sont rarement vides; ils sont généralement utilisés pour fournir une signification ou un contexte à une donnée, et en conséquence les éléments XML entourent le plus souvent des données. Chacun des éléments de la figure 2.1 est ouvert et fermé. Par exemple, le titre du document est dénoté par les éléments <title> et </title> et le seul paragraphe du document est dénoté par les éléments <p> et </p>. Les éléments ouvrants ne contiennent pas de slash, les éléments fermants en contiennent un.

Parfois des éléments peuvent être vides, comme la balise break en XHTML. Dans ces cas là la balise est ouverte et fermée à la fois, et elle est codée comme ceci:
.

Les éléments doivent toujours être correctement emboîtés

Des éléments XML ne peuvent être ouverts et fermés sans que les éléments qui ont été ouverts en dernier soient fermés. Ce serait un emboîtement incorrect. Prenez par exemple le marquage incorrect du paragraphe XHTML suivant:

```
<p>Ceci est un test. Ceci est un test du système d'<em>
<strong>alarme</em> sonore.</strong></p>
```

Dans l'exemple précédent, les éléments em et strong sont ouverts, mais l'élément em est fermé avant l'élément strong. Puisque l'élément strong a été ouvert après l'élément em il doit être fermé avant l'élément em. Voici le marquage correct:

```
<p>Ceci est un test. Ceci est un test du système d'<strong>
<em>alarme</em> sonore.</strong></p>
```

Les attributs d'éléments doivent toujours être entre

guillemets

Les éléments XML sont souvent qualifiés par l'utilisation d'attributs. Par exemple un nombre entier peut être désigné comme une longueur, et l'élément longueur peut être qualifié pour indiquer que l'unité de mesure est le pied. Par exemple: `<length unit='feet'>5</length>`. L'attribut est appelé unité, et sa valeur est toujours entre guillemets. On peut utiliser indifféremment des guillemets simples (') ou doubles (").

Il y a seulement cinq entités définies par défaut (<, >, &, ", et ')

En XML certains caractères ont une signification particulière, ce sont les caractères plus petit que (<), plus grand que (>), ainsi que le et commercial (&). Les deux premiers sont utilisés pour délimiter les noms d'éléments. Le et commercial est utilisé pour délimiter l'affichage de certains caractères spéciaux appelés entités; le et commercial est un caractè de "sortie" d'XML. Si vous voulez afficher ces trois caractères dans vos documents XML, vous devez les saisir sous leur forme d'entité:

- pour afficher le caractè & tapez `&`;
- pour afficher le caractère < tapez `<`;
- pour afficher le caractère > tapez `>`;

Les parseurs XML, des programmes qui traitent des documents XML, devraient être capables d'interpréter ces caractères sans qu'il soit besoin de les définir a priori.

Il y a deux autres caractères qui peuvent être représentés en tant que références d'entités:

- pour afficher le caractère ' vous pouvez taper `'`;
- pour afficher le caractère " vous pouvez taper `"`;

La sémantique d'XML

La sémantique d'un document XML (pensez grammaire) est une articulation des éléments XML qui existent dans un fichier, leurs relations, et leur signification. L'ironie, c'est que c'est là la partie délicate d'XML, qui a aboutie à une multitude de "langages" XML tels que: RSS, RDF, TEILite, DocBook, XMLMARC, EAD, XSL, etc. Dans le fichier XML suivant, qui est parfaitement valide, il y a un certain nombre d'éléments XML. Ce sont ces éléments qui donnent aux données leur valeur et leur signification:

```
<catalog>
  <work type='prose' date='1906'>
    <title>Les aventures du Far Ouest</title>
    <author>O Henry</author>
  </work>
  <work type='poetry' date='1845'>
    <title>Le corbeau</title>
    <author>Edgar Allen Poe</author>
  </work>
  <work type='play' date='1601'>
    <title>Hamlet</title>
    <author>William Shakespeare</author>
  </work>
```

</catalog>

Exercice - Vérifier la syntaxe XML

Dans cet exercice, vous apprendrez à identifier les erreurs syntaxiques dans un fichier XML.

1. Examinez le fichier suivant. Encerclez toutes ses erreurs syntaxiques, et corrigez les.

```
<name>Soupe d'huîtres</name>
<author>Eric Lease Morgan</author>
<copyright holder=Eric Lease Morgan>&copy; 2003</copyright>
<ingredients>
  <list>
    <item>1 branche de céleri
    <item>1 oignon
    <item>2 cuillères à café de beurre fondu
    <item>2 tasses d'huîtres dans leur jus
    <item>2 tasses de crème
  </list>
</ingredients>
<process>
  <P>Commencez par faire revenir le céleri et l'oignon dans le beurre pour les adoucir. Ajoutez l
  Chauffez jusqu'à ce que les huîtres flottent.
  Servir dans des bols chauds.</p>
  <p><i>Miam!</i></p></i>
</process>
```

Traduction possible des éléments: name=nom, author=auteur, copyright holder=détenteur du copyright, ingredients=ingrédients, list=liste, item=item, process=recette.

- A. Y a-t-il un et un seul élément racine?
- B. Les attributs sont-ils entre guillemets?
- C. Vérifiez l'utilisation d'entités invalides. Il y a deux erreurs dans ce fichier.
- D. Vérifiez que les balises ouvertes sont bien fermées. Cinq éléments ne sont pas fermés.
- E. Vérifiez l'emboîtement correct des éléments. Deux éléments ne sont pas correctement emboîtés.
- F. Vérifiez l'utilisation des casses dans les noms d'élément. Un élément n'utilise pas la bonne casse.

Chapter 3. Créer votre propre système de balises



But et éléments constitutifs

But et éléments constitutifs

Le "X" de XML signifie extensible. Les créateurs d'XML voulaient signifier par là qu'il serait facile de créer son propre système de balises -- un vocabulaire, ou un langage destiné à décrire un ensemble de données/informations. Il est important, pour créer un système de balises XML, d'abord de bien déterminer à quel usage est destiné le document, et ensuite de déterminer les éléments constitutifs d'un document et de leur assigner des éléments. Le processus par lequel on crée un système de balises XCML est similaire à celui par lequel on conçoit une application de base de données. Vous devez vous demander quelles données vous sont utiles, et trouver un endroit pour les enregistrer.

La création d'un système de balises pour une lettre est un excellent exemple:

11 Décembre 2002

Melville Dewey
Columbia University
New York, NY

Cher Melville,

J'ai lu vos textes concernant la nature de la bibliothéconomie, et je les trouve tout à fait fascinants. J'aimerais avoir l'occasion de discuter avec vous du rôle du catalogue papier dans les bibliothèques aujourd'hui, compte tenu du développement du World Wide Web. En particulier, dans quelle mesure des services comme Google ou Amazon.fr affectent-ils les demandes de nos lecteurs à l'égard des services de bibliothèques? M. Cutter et moi-même discuterons de ces sujets lors de la prochaine conférence annuelle, et nous serons disponibles aux dates et heures suivantes :

- * Lundi, 2-4
- * Mardi, 3-5
- * Jeudi, 1-3

Nous espérons que vous pourrez nous rejoindre.

Cordialement, S. R. Ranganathan

En lisant cette lettre, vous reconnaissez des sections communes à toutes les lettres. En analysant ces sections, il est possible de créer une liste d'éléments XML. Par exemple, cette lettre a une date, un bloc de texte définissant le destinataire, une salutation, un ou plusieurs paragraphes de texte, une liste, et une phrase de conclusion. A y regarder de plus près, il semble que certaines sections aient des sous-sections. Par exemple, le destinataire a un nom, une première ligne d'adresse, et une seconde ligne d'adresse. Par ailleurs, le corps du texte peut avoir une certaine mise en forme particulière.

La division en sections toujours plus petites pourrait aller jusqu'au mot individuel. Où s'arrêter? Ne créez des éléments que pour les données que vous allez réellement utiliser. Si vous n'avez jamais besoin de connaître la ville ou l'Etat de votre destinataire, alors ne créez pas d'élément correspondant. Posez-vous la question : quel est le but de ce document? Dans son contenu, quels éléments voulez-vous mettre en évidence? Si vous voulez créer une liste des villes dans lesquelles vous avez des correspondants, alors vous devrez créer un élément pour la ville. Si vous avez besoin d'extraire chaque phrase de votre document, alors il faudra aussi les délimiter par des balises. Sinon, faites des économies en temps et en énergie, restez simple.

Une fois que vous avez articulé les parties du document que vous voulez baliser, vous devez les nommer. Les noms d'éléments XML peuvent contenir les lettres A à Z et a à z, ainsi que les entiers 0 à 9. Ils peuvent aussi contenir des lettres accentuées et trois signes de ponctuation : underscore (_), tiret (-), et point (.). Les noms d'éléments ne peuvent pas contenir d'espace vide (espace, tabulation, retour chariot), ni d'autres signes de ponctuation. Autant être prudent : utilisez des lettres.

Il est temps maintenant de créer quelques éléments. En suivant ce qui précède nous pourrions créer en ensemble d'éléments tel que celui-ci (entre parenthèses des alternatives possibles en Français):

- letter (lettre)
- date (date)
- addressee (destinataire)
- name (nom)
- address_one (adresse_un)
- address_two (adresse_deux)
- greeting (salutation)
- paragraph (paragraphe)
- italics (italiques)
- list (liste)
- item (item)
- closing (conclusion)

En utilisant ces éléments pour cadre, on peut baliser le texte de la façon suivante:

```
<letter>
  <date>11 Décembre 2002</date>
```

```
<addressee>
  <name>Melville Dewey</name>
  <address_one>Columbia University</address_one>
  <address_two>New York, NY</address_two>
</addressee>

<greeting>Cher Melville,</greeting>

<paragraph>
J'ai lu vos textes concernant la nature de la bibliothéconomie, et <italics>je les trouve tout à fait
J'aimerais avoir l'occasion de discuter avec vous du rôle du catalogue papier dans les bibliothèques
aujourd'hui, compte tenu du développement du World Wide Web. En particulier, dans quelle mesure
des services comme Google ou Amazon.fr affectent-ils les demandes de nos lecteurs à l'égard
des services de bibliothèques? M. Cutter et moi-même discuterons de ces sujets lors de la prochaine
conférence annuelle, et nous serons disponibles aux dates et heures suivantes :</paragraph>

<list>
  <item>Lundi, 2-4</item>
  <item>Mardi, 3-5</item>
  <item>Jeudi, 1-3</item>
</list>

<paragraph>Nous espérons que vous pourrez nous rejoindre.</paragraph>

<closing>Cordialement, S. R. Ranganathan</closing>

</letter>
```

Exercice - Créez votre propre système de balises XML

Exercice - Créez votre propre système de balises XML

Dans cet exercice, vous allez créer votre propre système de balises XML, pour décrire une lettre simple.

1. Examinez la lettre suivante.

3 Février 2003

American Library Association
15 Huron Street
Chicago, IL 12304

A qui de droit:

J'ai cru comprendre que l'Association ne compte plus consacrer de budget à la réalisation d'affiches de personnalité vantant les mérites de la lecture. C'est quoi au juste votre problème! Ne savez-vous pas que la lecture est fondamentale? Ces affiches nous aident vraiment, moi et aussi mes usagers. Je les trouvais super.

S'il-vous-plaît veuillez reconsidérer votre position.

Cordialement, B. Ig Reeder

2. Collectivement, décidez des éléments qui peuvent être utilisés pour baliser cette lettre et en faire un document

XML.

- A. Quel peut être l'élément racine?
 - B. Quelles sont les parties qui constituent cette lettre? Quels noms d'éléments peut-on donner à ces parties?
 - C. Certaines de ces parties, comme l'adresse ou la salutation, ont des sous-parties. Comment appellerons-nous des sous-parties?
 - D. Avec un stylo ou un crayon, balisez la lettre ci-dessus en utilisant les éléments convenus.
3. Balisez la lettre en XML, et validez sa syntaxe en utilisant un navigateur web.
- A. Utilisez le bloc-notes pour ouvrir le fichier ala.txt
 - B. Ajoutez l'élément racine au début et à la fin de du fichier.
 - C. Balisez chaque partie et sous-partie de la lettre en utilisant les noms d'éléments convenus.
 - D. Sauvegardez le fichier sous le nom ala.xml.
 - E. Ouvrez ala.xml dans votre navigateur web, et corrigez les éventuelles erreurs signalées. S'il n'y a pas d'erreurs, alors félicitation, vous venez de baliser votre premier document XML.

Chapter 4. Gérer l'affichage d'XML avec les feuilles de style



Introduction

Les feuilles de style en cascade (Cascading Style Sheets - CSS) peuvent être utilisées pour restituer des documents XML dans une forme plus lisible pour l'homme. Les fichiers CSS permettent de séparer la présentation du contenu.

Les CSS se composent de trois parties : la mise en page, la typographie, et la couleur. En associant un fichier XML à un fichier CSS et en les affichant dans un navigateur, il est possible d'afficher le contenu du document XML sous une forme esthétiquement satisfaisante.

Les fichiers CSS sont constitués de "sélecteurs" et "déclarations". Chaque sélecteur dans un fichier CSS correspond à un élément dans un fichier XML. Chaque sélecteur est ensuite constitué de déclarations -- des paires standardisées : nom/valeur -- qui indiquent la façon dont le contenu des éléments XML doit être affiché. Ça ressemble à quelque chose comme ça : `note { display: block; }`.

Voici un document XML très simple décrivant une note:

```
<?xml-stylesheet type="text/css" href="note.css"?>
<note>
  <para>Notes are very brief documents (Les notes sont des documents courts).</para>
  <para>They do not contain very much content (qui ne contiennent pas grand chose).</para>
</note>
```

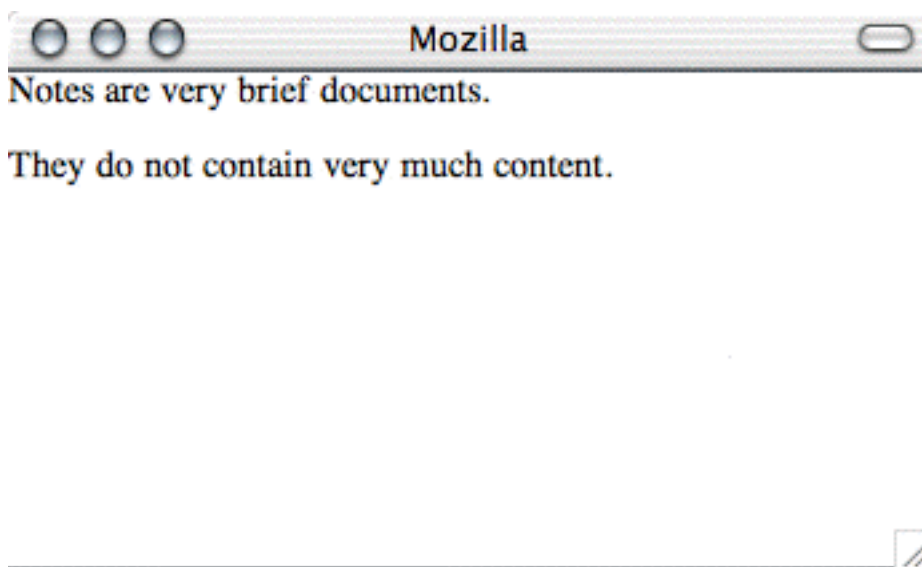
La première chose que vous remarquerez à propos de ce document, c'est la toute première ligne, qui est une instruction de traitement XML. Cette instruction particulière indique à l'application qui lit le document XML, de l'afficher en utilisant un fichier CSS intitulé `note.css`. La symétrie du document XML devrait vous être par ailleurs familière.

Si je voulais afficher le contenu de la note de telle façon que chaque paragraphe soit séparé par une ligne blanche, le fichier CSS devrait ressembler à quelque chose comme ça :

```
note { display: block; }  
para { display: block; margin-bottom: 1em; }
```

Dans ce fichier CSS il y a deux sélecteurs qui correspondent aux éléments du fichier XML : note et para. Chaque sélecteur est associé à une ou plusieurs paires nom/valeur (déclarations) qui décrit la façon dont le contenu des éléments doit être affiché. Chaque nom est séparé de la valeur par le signe deux-points (:), les paires de noms/valeurs sont séparées les unes des autres par un point virgule (;), et toutes les déclarations associées à un sélecteur sont regroupées dans des accolades ({}).

Si vous ouvrez note.xml dans un navigateur web relativement moderne, ça devrait donner quelque chose comme ça :



Attention, tous les navigateurs ne gèrent pas les feuilles de style de la même façon. (Quelle surprise!) En général, vous aurez des résultats moins bon avec Netscape Navigator 4.7 et Internet Explorer 5.0. Mozilla 1.0 et Internet Explorer 6.0 gèrent beaucoup mieux les feuilles de style.

Pour pouvoir utiliser des fichiers CSS, il faut surtout savoir créer les déclarations par paires nom/valeur. Pour une liste complète de ces paires nom/valeur, voir la description de CSS par le World Wide Web Consortium. [<http://www.w3.org/TR/REC-CSS2/propidx.html>] . Un certain nombre de paires sont décrites ci-dessous.

display

La propriété display (affichage) est utilisée pour spécifier si un élément doit ou non être affiché; et si oui, comment (mais cela de façon très générale seulement). Les principales valeurs pour display sont : inline, block, list-item, ou none. Inline est la valeur par défaut. Cela signifie que le contenu d'un élément ne comprendra pas de retour à la ligne après le contenu; le contenu sera affiché comme une simple ligne de texte. Attribuer la valeur block à la propriété display crée un retour à la ligne après le contenu de l'élément. Pensez aux blocks comme à des paragraphes. La valeur list-item est similaire à block, sauf qu'elle met légèrement le texte en retrait. L'usage de la valeur none signifie que le contenu ne sera pas affiché; le contenu est caché. Exemples:

- display: none;
- display: inline;
- display: block;

- `display: list-item;`

margin

La propriété `margin` (marge) est utilisée pour indiquer la taille de l'espace blanc qui entoure les blocs de texte. Les valeurs peuvent être des pourcentages (%), des pixels (px), ou des conventions typographiques traditionnelles telles que l'unité `em` (em). Quand on donne une valeur à la propriété `margin`, elle s'applique simultanément aux marges droite, gauche, haut et bas. Il est possible de spécifier différentes marges en utilisant les propriétés `margin-top`, `margin-bottom`, `margin-left`, et `margin-right`. Exemples:

- `margin: 5%;`
- `margin: 10px;`
- `margin-top: 2em;`
- `margin-left: 85%;`
- `margin-right: 50px;`
- `margin-bottom: 1em;`

text-indent

Comme la propriété `margin`, la propriété `text-indent` accepte pour valeurs les pourcentages, les pixels, ou les unités typographiques. Cette propriété est utilisée pour définir la façon dont certaines lignes doivent être mises en retrait dans des blocs de texte. Par exemple:

- `text-indent: 2em;`
- `text-indent: 3%;`

text-align

Les valeurs courantes de `text-align` sont `right`, `left`, `center`, et `justify`. Elles sont utilisées pour aligner le texte à l'intérieur d'un bloc de texte. Ces valeurs fonctionnent de la même façon que votre traitement de texte. Par exemple:

- `text-align: right;`
- `text-align: left;`
- `text-align: center;`
- `text-align: justify;`

list-style

Les listes à puces sont faciles à lire et d'usage courant dans la présentation typographique d'aujourd'hui. Si vous

voulez créer une liste, vous utiliserez d'abord le sélecteur `display: list-item` pour la liste en tant que telle, puis quelque chose comme `disc`, `circle`, `square`, ou `decimal` pour la valeur `list-style`. Par exemple:

- `list-style: circle;`
- `list-style: square;`
- `list-style: disc;`
- `list-style: decimal;`

font-family

Associez `font-family` à un sélecteur si vous voulez décrire dans quelle police de caractère doit être affichée le XML. Parmi les valeurs, on trouve les noms des polices ainsi qu'un certain nombre de familles de polices génériques telles que `serif` ou `sans-serif`. Les noms de familles de polices qui contiennent plus d'un mot doivent être placés entre guillemets simples. Par exemple:

- `font-family: helvetica;`
- `font-family: times, serif;`
- `font-family: 'cosmic cartoon', sans-serif;`

font-size

Les tailles des polices peuvent être indiquées avec des dimensions exactes en points tout autant qu'avec des tailles relatives telles que `small`, `x-small`, ou `large`. Par exemple:

- `font-size: 12pt;`
- `font-size: small;`
- `font-size: x-small;`
- `font-size: large;`
- `font-size: xx-large;`

font-style

Les valeurs habituelles pour le style de police sont `normal` ou `italic`; elles indiquent la façon dont le texte s'affiche, comme par exemple dans :

- `font-style: normal;`
- `font-style: italic;`

font-weight

Ce paramètre est utilisé pour indiquer si le texte doit être rendu en gras ou non. Les valeurs habituelles sont normal et bold:

- font-weight: normal;
- font-weight: bold;

Rassembler tout ça

Vous trouverez ci-dessous un fichier CSS à appliquer au fichier letter.xml que nous avons déjà vu. Notez que chaque élément du fichier XML a un sélecteur correspondant dans le fichier XML. Pour indiquer à votre navigateur web d'utiliser le fichier CSS, vous devrez ajouter l'instruction de traitement de la feuille de style (<?xml-stylesheet type="text/css" href="letter.css" ?>) au début de letter.xml.

```
letter {
    display: block;
    margin: 5%
}

date, addressee {
    display: block;
    margin-bottom: 1em
}

name, address_one, address_two { display: block }

greeting, list {
    display: block;
    margin-bottom: 1em;
}

paragraph {
    display: block;
    margin-bottom: 1em;
    text-indent: 1em
}

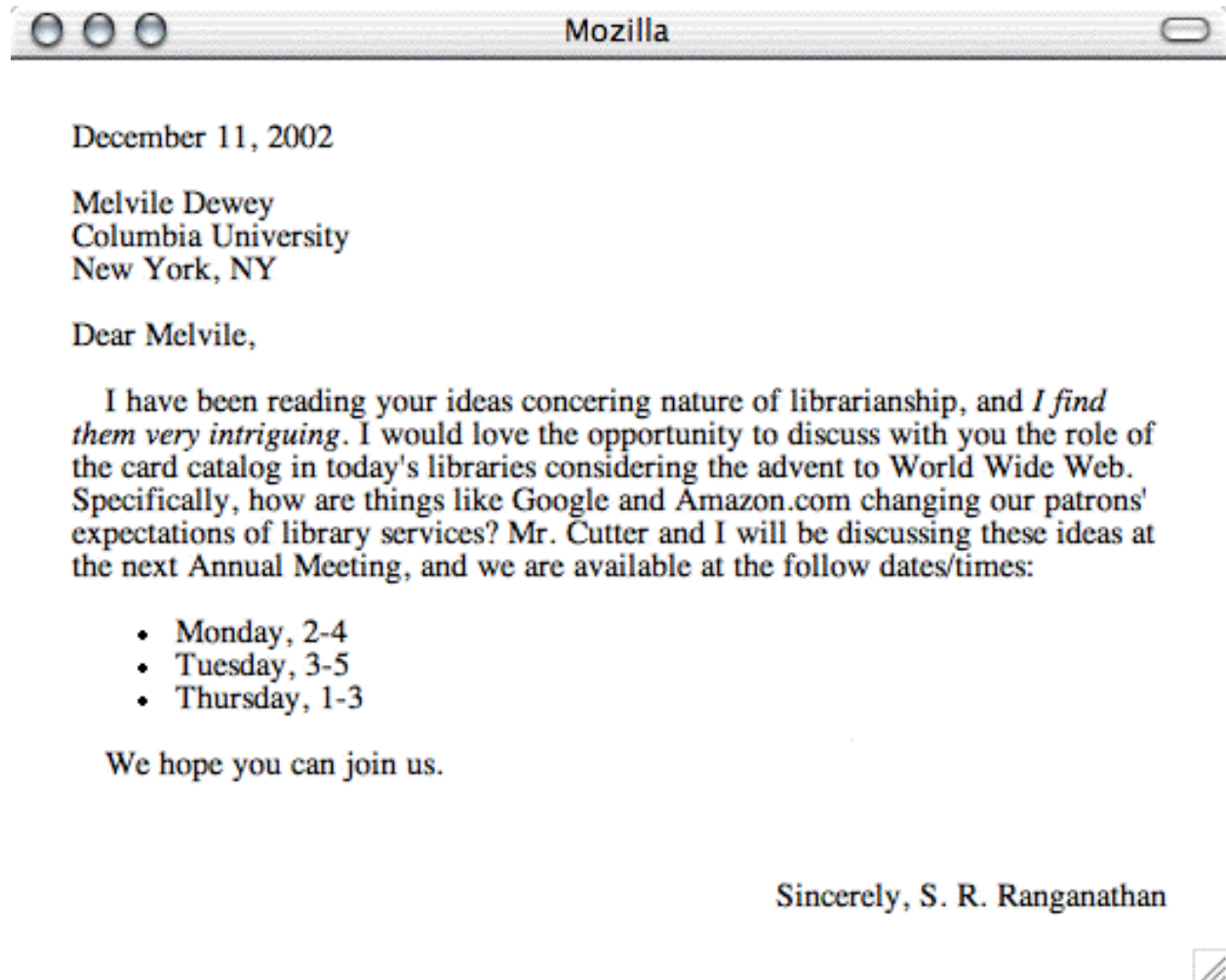
italics {
    display: inline;
    font-style: italic
}

list { display: block }

item {
    display: list-item;
    list-style: inside;
    text-indent: 2em
}

closing {
    display: block;
    margin-top: 3em;
    text-align: right
}
```

Le fichier XML correspondant devrait ressembler à quelque chose comme ça:



Tables

Les tableaux (tables) sont des listes à double entrée, qui se présentent sous la forme d'un canevas de lignes et de colonnes. Une liste d'ouvrages, très simple (un catalogue) se prête bien à un affichage en tables puisque dans cette liste chaque livre (oeuvre) possède un certain nombre d'attributs tels que le titre, l'auteur, le genre, la date. Chaque oeuvre représente une ligne, et le titre, l'auteur, le genre et la date représentent les colonnes.

Voici un fichier XML représentant un catalogue rudimentaire. Notez que l'instruction de traitement XML indique au logiciel de traitement qu'il doit afficher le contenu de ce fichier en utilisant un fichier CSS appelé catalog.css :

```
<?xml-stylesheet href='catalog.css' type='text/css'?>
<catalog>
  <caption>Ceci est mon catalogue personnel.</caption>
  <structure>
    <title>Titre</title>
    <author>Auteur</author>
    <type>Genre</type>
    <date>Date</date>
  </structure>
  <work>
    <title>Les aventures du Far Ouest</title>
    <author>O Henry</author>
```

```
<type>prose</type>
<date>1906</date>
</work>
<work>
  <title>Le corbeau</title>
  <author>Edgar Allen Poe</author>
  <type>prose</type>
  <date>1845</date>
</work>
<work>
  <title>Hamlet</title>
  <author>William Shakespeare</author>
  <type>prose</type>
  <date>1601</date>
</work>
</catalog>
```

CSS permet de gérer les tableaux, mais encore une fois les navigateurs actuels ne savent pas tous traiter efficacement les tableaux. Pour créer un tableau vous devez apprendre au moins trois nouvelles valeurs qui s'attachent à l'élément display:

1. display: table;
2. display: table-row;
3. display: table-cell;

Si nous utilisons l'exemple du catalogue ci-dessus, display: table sera associé à l'élément catalog, display: table-row sera associé à l'élément work, et display: table-cell sera associé aux éléments title, author, type, et date.

Par ailleurs, vous pouvez vouloir rendre vos tableaux à la fois plus complets et plus lisibles:

1. display: table-caption;
2. display: table-header-group;

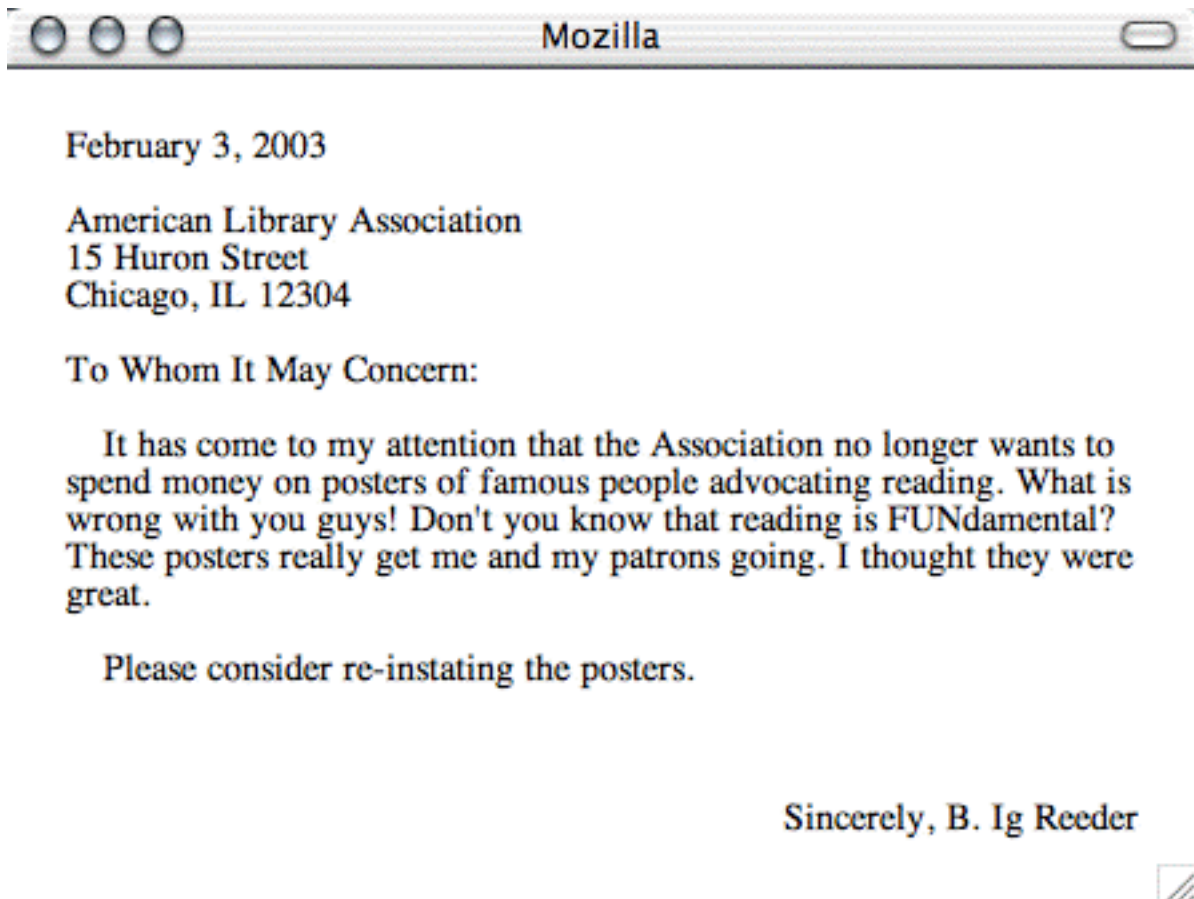
Table-caption est utilisé pour donner un titre d'ensemble au tableau. Table-header-group est utilisé pour donner un titre aux colonnes.

Exercice - Afficher du XML avec CSS

Dans cet exercice, vous apprendrez à écrire un fichier CSS et à l'utiliser pour l'affichage d'un fichier XML.

1. Créez un fichier CSS permettant l'affichage du fichier ala.xml créé dans un exercice précédent.
 - A. Ouvrez ala.xml dans un bloc-notes.
 - B. Ajoutez l'instruction de traitement XML `<?xml-stylesheet href="ala.css" type="text/css"?>` au début du fichier. Enregistrez-le.
 - C. Créez un nouveau fichier vierge dans le bloc-notes et enregistrez le avec le nom ala.css.
 - D. Dans ala.css, faites la liste de tous les éléments XML que contient ala.xml en mettant un élément par ligne.

- E. Assignez à chaque élément un sélecteur display accompagné d'une valeur block (ex. `para { display: block; }`).
 - F. Ouvrez `ala.xml` dans votre navigateur web pour vérifier vos progrès.
 - G. Ajoutez une ligne entre chaque partie de la lettre en ajoutant `margin-bottom: 1em` à chaque sélecteur au début de chaque partie (ex. `para { display: block; margin-bottom: 1em; }`).
 - H. Ouvrez `ala.xml` dans votre navigateur web pour vérifier vos progrès.
 - I. Changez le sélecteur display utilisé pour la salutation de façon à ce que son contenu s'affiche non pas comme un bloc mais en retrait (ex. `salutation { display: inline; }`).
 - J. Ouvrez `ala.xml` dans votre navigateur web pour vérifier vos progrès.
2. Mettez les paragraphes en retrait en ajoutant `text-indent: 2em` à l'élément `para`. Le résultat final devrait ressembler à quelque chose comme ça :



Chapter 5. Transformer XML avec XSLT



Introduction

En dehors des fichiers CSS, il existe une autre méthode pour transformer des documents XML en quelque chose de plus facile à lire. Il s'agit du langage de feuilles de style extensible: transformation (eXtensible Stylesheet Language: Transformation, ou XSLT). XSLT est un langage de programmation implémenté comme sémantique XML. Comme avec CSS, vous écrivez/créez d'abord un fichier XML, puis vous écrivez un fichier XSLT et vous utilisez un programme informatique pour combiner les deux afin de produire un troisième fichier. Le troisième fichier peut être un fichier plein texte contenant un autre fichier XML, un texte narratif, ou même un ensemble de commandes sophistiquées, comme des requêtes SQL (Structured Query Language) qui seront appliquées à un gestionnaire de bases de données.

A la différence de CSS ou XHTML, XSLT est un langage de programmation. Un langage complet, avec des paramètres, des processus conditionnels, et des appels de fonctions. A la différence de la plupart des langages de programmation, XSLT est déclaratif, et non procédural. Cela signifie que le programme n'est pas exécuté de la première à la dernière ligne, mais quand le programme XSLT rencontre une donnée, il l'évalue et exécute en conséquence les parties du programme XSLT requises: les programmes XSLT ne sont pas procéduraux. Cela signifie aussi qu'il n'est pas possible de changer les valeurs des variables une fois qu'elles ont été définies.

Il y a un certain nombre de processeurs XSLT disponibles pour diverses plate-formes spécifiques en Java, Perl, ou en fonction des systèmes d'exploitation:

- Xerces et Xalan [<http://xml.apache.org/>] - implémentations Java
- xsltproc [<http://xmlsoft.org/XSLT/xsltproc2.html>] - une application binaire qui utilise diverses bibliothèques en C, et qui est aussi fournie avec un programme qui s'appelle xmllint, utilisé pour valider les documents XML
- Sablotron [http://www.gingerall.com/charlie/ga/xml/p_sab.xml] - une autre distribution binaire qui utilise les bibliothèques C++ et qui a une API Perl et une API Python
- Saxon [<http://saxon.sourceforge.net/>] - une autre implémentation Java

Afficher du texte

Dans cette section nous allons transformer une lettre, créée précédemment, en un fichier XHTML. Nous créerons d'abord un fichier XSLT en utilisant diverses commandes XML, puis nous combinerons le fichier XSLT avec notre lettre en utilisant un processeur XSLT. Voici une liste des diverses commandes XSLT que nous utiliserons. Souvenez-vous que XSLT est un langage de programmation qui apparaît sous la forme d'un fichier XML. Donc, chacune des commandes est un élément XML, et les commandes sont qualifiées à l'aide d'attributs XML.

- **stylesheet** - C'est la racine de tous les fichiers XSLT. Elle a besoin d'attributs qui définissent le nom et la version de XSLT. Voici la définition à peu près standard de XSLT: `<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">`.
- **output** - est utilisé pour dénoter le type de fichier texte qui sera créé en sortie, et si ce résultat doit ou non être indenté et/ou se voir appliquer une DTD. Par exemple, l'usage suivant de output dit au processeur XSLT d'indenter le fichier résultat pour rendre le texte plus facile à lire : `<xsl:output indent="yes" />`.
- **template** - Cette commande est utilisée pour chercher un endroit particulier d'un document XML. Elle requiert l'usage d'un attribut appelé match, et est utilisée pour dénoter la branche de l'arborescence XML qui doit être traitée. Par exemple, l'usage suivant de template identifie l'élément racine du document XML : `<xsl:template match="/">`.
- **value-of** - est utilisé pour récupérer le résultat de l'attribut obligatoire appelé select, qui définit exactement ce qui doit être récupéré. Dans cet exemple, le processeur XSLT produira en sortie la valeur de l'élément date d'une lettre : `<xsl:value-of select="/letter/date/" />`.
- **apply-templates** - recherche dans le fichier XSLT en cours un template défini dans la déclaration de la commande select, ou bien produit comme résultat le contenu du noeud en cours du fichier XML s'il n'y a pas de template correspondant. Ici la commande apply-templates indique au processeur qu'il doit trouver dans le fichier XSLT en cours quelque chose correspondant aux éléments paragraph ou list : `<xsl:apply-templates select="paragraph | list" />`.
- En plus des commandes XSLT (les éléments), les fichiers XSLT peuvent contenir du texte et/ou des balises XML. Quand il rencontre ce texte ou ces balises, le processeur XSLT doit simplement afficher ces valeurs. C'est ce qui nous permet de créer une version XHTML. Le processeur lit le fichier XML et le fichier XSLT. En lisant le fichier XSLT, il applique les commandes XSLT ou affiche les valeurs des commandes non-XSLT pour produire un autre fichier XML, ou un autre fichier texte.

Vous trouverez ci-dessous notre premier exemple XSLT. Il est destiné à être appliqué au fichier letter.xml, pour produire un fichier valide en XHTML. Vous pouvez le voir fonctionner en utilisant un processeur nommé xsltproc. En partant du principe que tous les fichiers nécessaires se trouvent dans le même répertoire, la commande xsltproc est **xsltproc -o letter.html letter2html.xsl letter.xml**.

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

<!-- letter2html.xsl; un fichier XSL -->

<!-- défini le résultat comme étant un fichier XML, en l'occurrence, un fichier XHTML -->
<xsl:output
method="xml"
omit-xml-declaration="no"
indent="yes"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" />

<!-- démarre à la racine du fichier letter -->
<xsl:template match="letter">

  <!-- affiche un élément racine XHTML -->
```

```
<html>

  <!-- ouvre l'élément head de XHTML -->
  <head>

    <!-- affiche un élément titre avec le nom du destinataire -->
    <title><xsl:value-of select="addressee/name"/></title>

  <!-- ferme l'élément head -->
  </head>

  <!-- ouvre la balise body et lui attribue un style -->
  <body style="margin: 5%">

    <!-- trouve diverses templates dans le fichier XSLT avec les valeurs qui leur sont associées -->
    <xsl:apply-templates select="date"/>
    <xsl:apply-templates select="addressee"/>
    <xsl:apply-templates select="greeting"/>
    <xsl:apply-templates select="paragraph | list" />
    <xsl:apply-templates select="closing"/>

  <!-- ferme la balise body -->
  </body>

</html>

</xsl:template>

<!-- date -->
<xsl:template match="date">

  <!-- affiche une balise de paragraphe et le contenu du noeud en cours, en l'occurrence date -->
  <p><xsl:apply-templates/></p>

</xsl:template>

<!-- destinataire -->
<xsl:template match="addressee">

  <!-- ouvre un paragraphe -->
  <p>

    <!-- affiche le contenu des éléments name, address_one,
    et address_two du fichier letter.xml, ainsi que deux balises br -->
    <xsl:value-of select="name"/><br />
    <xsl:value-of select="address_one"/><br />
    <xsl:value-of select="address_two"/>

  <!-- ferme le paragraphe -->
  </p>

</xsl:template>

<!-- chacune des templates suivante fonctionne exactement de la même façon que la template date -->

<!-- greeting -->
<xsl:template match="greeting">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<!-- paragraphe -->
<xsl:template match="paragraph">
  <p style="text-indent: 1em">
    <xsl:apply-templates/>
  </p>
</xsl:template>

<!-- conclusion -->
```

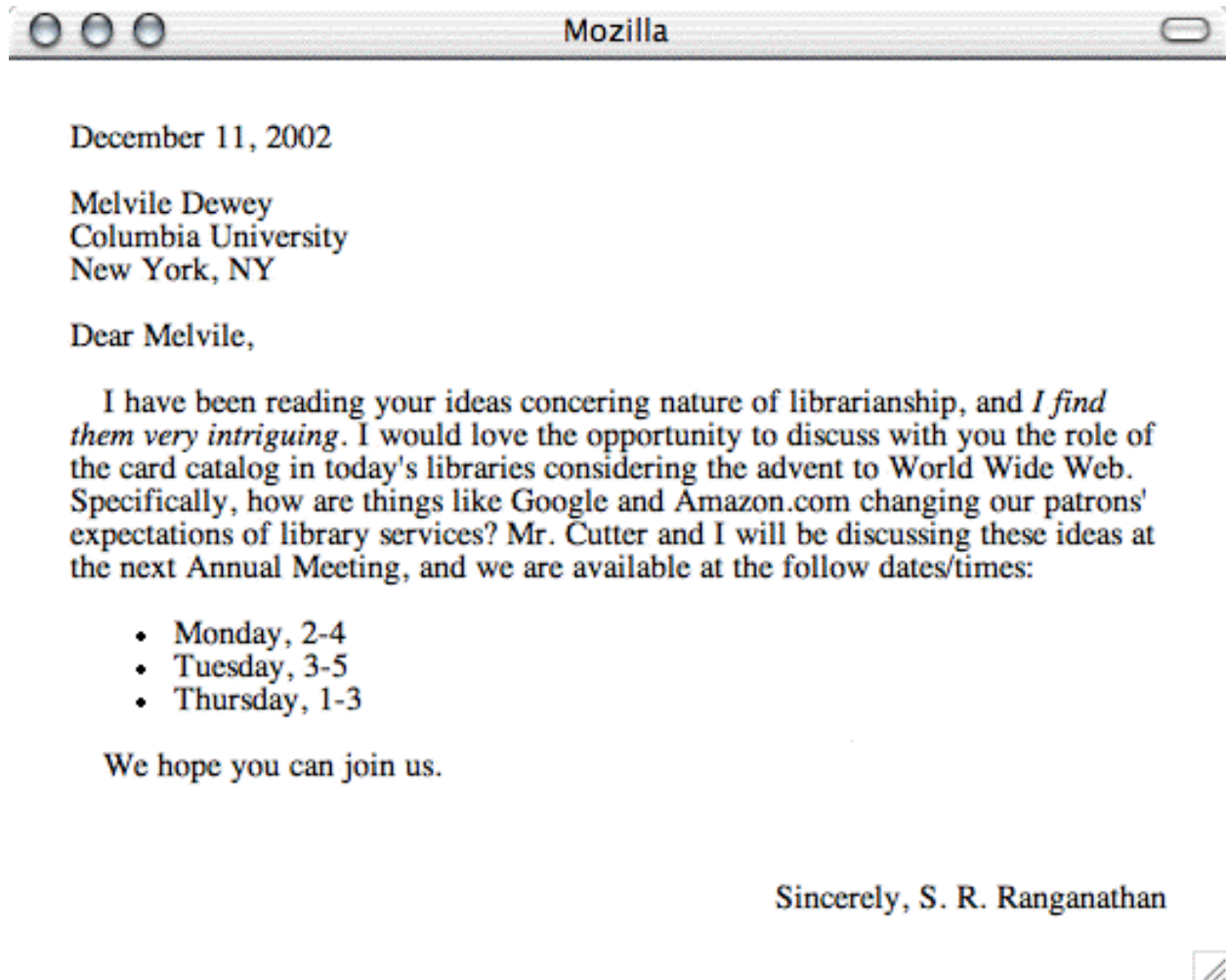
```
<xsl:template match="closing">
  <p style="margin-top: 3em; text-align: right">
    <xsl:apply-templates/>
  </p>
</xsl:template>

<!-- italiques -->
<xsl:template match="italics">
  <i>
    <xsl:apply-templates/>
  </i>
</xsl:template>

<!-- liste -->
<xsl:template match="list">
  <ul>
    <xsl:apply-templates/>
  </ul>
</xsl:template>

<!-- item -->
<xsl:template match="item">
  <li>
    <xsl:apply-templates/>
  </li>
</xsl:template>
</xsl:stylesheet>
```

Le résultat final devrait ressembler à ça:



Evidemment, l'exemple ci-dessus semble compliqué et fonctionne exactement comme nos fichiers CSS. En même temps, afficher le fichier letter.xml avec CSS nécessite un navigateur web récent. Si le fichier letter2html.xsl était incorporé dans un serveur web, le navigateur web n'aurait pas besoin de comprendre CSS. Etant donné l'exemple ci-dessus rien ne nous impose d'utiliser XSLT, pour l'instant.

Afficher des données tabulées

Voici un autre exemple de fichier XSLT utilisé pour afficher un fichier XML. Cet exemple affiche notre fichier catalog.xml. Lui aussi fonctionne largement comme un bon vieux fichier CSS. Vous pouvez le transformer en utilisant xsltproc comme ceci: `xsltproc -o catalog.html catalog2html.xsl catalog.xml`.

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

  <!-- catalog2html.xsl -->

  <xsl:output
    method="xml"
    omit-xml-declaration="no"
    indent="yes"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" />

<!-- catalog -->
<xsl:template match="catalog">
  <html>
    <head>
      <title><xsl:value-of select="caption"/></title>
    </head>
    <body>
      <table>
        <xsl:apply-templates select="caption"/>
        <xsl:apply-templates select="structure"/>
        <xsl:apply-templates select="work"/>
      </table>
    </body>
  </html>
</xsl:template>

<!-- en-tête -->
<xsl:template match="caption">
  <caption style="text-align: center; margin-bottom: 1em">
    <xsl:value-of select="."/>
  </caption>
</xsl:template>

<!-- structure -->
<xsl:template match="structure">
  <thead style="font-weight: bold">
    <tr><xsl:apply-templates/></tr>
  </thead>
</xsl:template>

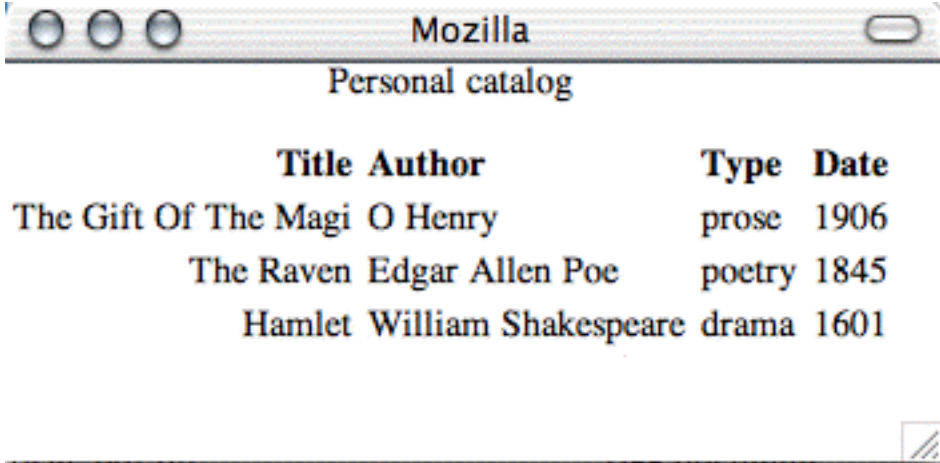
<!-- oeuvre -->
<xsl:template match="work">
  <tr><xsl:apply-templates/></tr>
</xsl:template>

<!-- titre -->
<xsl:template match="title">
  <td style="text-align: right; padding: 3px"><xsl:value-of select="."/></td>
</xsl:template>

<!-- auteur, genre, ou date -->
<xsl:template match="author | type | date">
  <td><xsl:value-of select="."/></td>
</xsl:template>

</xsl:stylesheet>
```

A nouveau, le résultat final devrait ressembler à ça:



Title	Author	Type	Date
The Gift Of The Magi	O Henry	prose	1906
The Raven	Edgar Allen Poe	poetry	1845
Hamlet	William Shakespeare	drama	1601

Manipuler des données XML

Les fichiers CSS, comme les fichiers XSLT ci-dessus, traitent les fichiers XML du début à la fin. Cette technique n'utilise pas les avantages des caractéristiques programmatiques de XSLT. Ce que fait par contre l'exemple suivant. D'abord l'exemple suivant va chercher une valeur qu'il utilise pour faire un tri (sort by). Ensuite, ce fichier XSLT utilise quelques appels de fonctions comme count, sum et sort. Et là réside une distinction importante entre CSS et XSLT. CSS ne sert qu'à l'affichage. XSLT peut être utilisé pour afficher du contenu XML. Mais il peut aussi être utilisé pour manipuler des contenus.

Dans cet exemple on fait des calculs en utilisant notre liste d'animaux. D'abord on affiche le total du nombre d'animaux, ainsi que leur âge moyen. Ensuite la liste des animaux peut être triée par le nom, l'âge, l'espèce, ou la couleur. Pour en voir la démonstration, essayez la commande suivante : **xsltproc -o pets.html --stringparam sortby age pets2html.xml pets.xml** . Différents résultats peuvent être obtenus en changeant la valeur de sortby de name à color ou type. Que se passe-t-il si une valeur sortby erronée est transmise au fichier XSLT? Que se passe-t-il si aucun valeur stringparam n'est transmise? Pourquoi?

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

  <!-- pets2html.xml -->

  <xsl:output
    method="xml"
    omit-xml-declaration="no"
    indent="yes"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" />

  <!-- obtient un paramètre et le sauvegarde dans une variable appelée sortby; utilise name par défaut -->
  <xsl:param name="sortby" select="'name'"/>

  <!-- animaux -->
  <xsl:template match="pets">

    <html>
      <head>
        <title>Pets</title>
      </head>
      <body style="margin: 5%">
        <h1>Pets</h1>
        <ul>

          <!-- utilise la fonction count pour déterminer le nombre d'animaux -->
```

```
<li>Total number of pets: <xsl:value-of select="count(pet)"/></li>

<!-- calcule l'âge moyen des animaux en utilisant les fonctions sum et count, ainsi que l'opéra
<li>Average age of pets: <xsl:value-of select="sum(pet/age) div count(pet)"/></li>
</ul>
<p>Pet sorted by: <xsl:value-of select="$sortBy"/></p>
<table>
  <thead>
    <tr>
      <td style="text-align: right; font-weight: bold">Name</td>
      <td style="text-align: right; font-weight: bold">Age</td>
      <td style="font-weight: bold">Type</td>
      <td style="font-weight: bold">Color</td>
    </tr>
  </thead>
  <xsl:apply-templates select="pet">

    <!-- Tri les animaux par un sous-élément particulier ($sortBy) -->
    <xsl:sort select="*[name()=$sortBy]"/>
  </xsl:apply-templates>
</table>
</body>
</html>
</xsl:template>

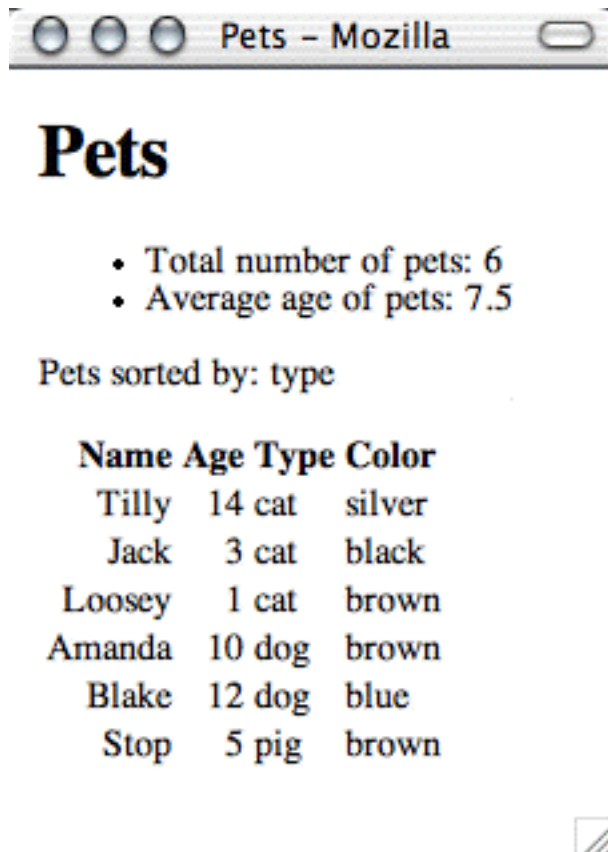
<!-- animal -->
<xsl:template match="pet">
  <tr><xsl:apply-templates/></tr>
</xsl:template>

<!-- name -->
<xsl:template match="name">
  <td style="text-align: right"><xsl:value-of select="."/></td>
</xsl:template>

<!-- âge -->
<xsl:template match="age">
  <td style="text-align: right"><xsl:value-of select="."/></td>
</xsl:template>

<!-- espèce ou couleur -->
<xsl:template match="type | color">
  <td><xsl:value-of select="."/></td>
</xsl:template>
</xsl:stylesheet>
```

Ce qui donne:



Utiliser XSLT pour créer d'autres types de fichiers textuels

Ce dernier exemple utilise encore le fichier `pets.xml`. Cette fois le fichier XSLT est utilisé pour un autre type de résultat, en l'occurrence un ensemble très simple de requêtes SQL. Le but est d'illustrer la façon dont le fichier `pets.xml` peut recevoir un nouvel usage. Une fois pour l'affichage, une fois pour le stockage. Utilisez cette commande pour voir le résultat : `xsltproc -o pets.sql pets2sql.xsl pets.xml`.

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

  <!-- pets2sql.xsl -->

  <!-- créé un affichage plein texte -->
  <xsl:output method="text" />

  <!-- trouve chaque animal -->
  <xsl:template match="pets">

    <!-- fait une boucle pour chaque animal -->
    <xsl:for-each select="pet">

      <!-- affiche une requête SQL INSERT pour chaque animal -->
      INSERT INTO pets (name, age, type, color)
      WITH VALUES ('<xsl:value-of select="name" />',
        '<xsl:value-of select="age" />',
        '<xsl:value-of select="type" />',
```

```
'<xsl:value-of select="color" />');  
</xsl:for-each>  
</xsl:template>  
</xsl:stylesheet>
```

Voici le SQL créé par le fichier XSLT ci-dessus:

```
INSERT INTO pets (name, age, type, color) WITH VALUES ('Puce', '14', 'chat', 'argent');  
INSERT INTO pets (name, age, type, color) WITH VALUES ('Amanda', '10', 'chien', 'marron');  
INSERT INTO pets (name, age, type, color) WITH VALUES ('Jack', '3', 'chat', 'noir');  
INSERT INTO pets (name, age, type, color) WITH VALUES ('Bernard', '12', 'chien', 'bleu');  
INSERT INTO pets (name, age, type, color) WITH VALUES ('Naia', '1', 'chat', 'marron');  
INSERT INTO pets (name, age, type, color) WITH VALUES ('Stop', '5', 'cochon', 'marron');
```

Ce fichier pourrait être fourni à un programme de base de données et remplir une table de données.

Cette section effleure seulement XSLT. C'est un langage de programmation en soi et les promesses d'XML résident dans l'exploitation de XSLT pour générer divers types de résultats, que ce soit pour les navigateurs web, les bases de données, ou comme données pour d'autres programmes informatiques.

Chapter 6. Les Document Type Definitions (DTD)



Définir des vocabulaires XML avec les DTDs

C'est une chose de créer votre propre marquage XML, mais si vous voulez partager vos documents avec d'autres, vous devrez leur communiquer le vocabulaire qui est compris par vos documents XML. C'est la partie sémantique des documents XML -- Quels éléments contiennent vos documents XML, et comment ces éléments sont-ils liés les uns aux autres? Ces relations sémantiques sont créées par les Document Type Definitions (DTD) et/ou des schémas XML. Les DTD sont un héritage du monde SGML. Elles sont plus couramment utilisées que les Schémas XML, plus nouveaux, basés sur XML. Cette section propose une vue d'ensemble de la création de DTD.

Une DTD peut exister dans ou en dehors d'un document XML. Si elle est présente dans le document XML, alors elle commence par une déclaration DOCTYPE, suivie du nom de l'élément racine du document XML, et enfin par une liste de tous les éléments et de leurs relations. Voici une DTD simple, incluse dans le fichier pets.xml lui-même:

```
<!DOCTYPE pets [  
  <!ELEMENT pets    (pet+)>  
  <!ELEMENT pet     (name, age, type, color)>  
  <!ELEMENT name    (#PCDATA)>  
  <!ELEMENT age     (#PCDATA)>  
  <!ELEMENT type    (#PCDATA)>  
  <!ELEMENT color   (#PCDATA)>  
  
<pets>  
  <pet>  
    <name>Puce</name>  
    <age>14</age>  
    <type>chat</type>  
    <color>argent</color>  
  </pet>  
  <pet>  
    <name>Amanda</name>  
    <age>10</age>  
    <type>chien</type>  
    <color>marron</color>  
  </pet>  
  <pet>  
    <name>Jack</name>  
    <age>3</age>
```

```

    <type>chat</type>
    <color>marron</color>
  </pet>
</pet>
  <name>Bernard</name>
  <age>12</age>
  <type>chien</type>
  <color>bleu</color>
</pet>
</pet>
  <name>Naia</name>
  <age>1</age>
  <type>chat</type>
  <color>Marron</color>
</pet>
</pet>
  <name>Puce</name>
  <age>5</age>
  <type>cochon</type>
  <color>marron</color>
</pet>
</pets>

```

Plus couramment, une DTD est placée en dehors du document XML, dans la mesure où elle sera utilisée par plusieurs documents XML. Dans ce cas, la déclaration DOCTYPE comprend un lien vers le fichier dans lequel les éléments XML sont décrits.

```

<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">

```

Que la DTD soit interne ou externe, une liste des éléments XML doit être clairement définie. Chaque élément de la liste ressemblera à quelque chose comme `!ELEMENT pets (pet+)` où `!ELEMENT` dénote un élément, `"pets"` est l'élément défini, et `"(pet+)"` est la définition. Ce sont les définitions qui constituent la partie délicate. Les définitions peuvent inclure bien des valeurs, et seules quelques unes d'entre elles sont décrites ici.

Noms et nombres d'éléments

Pour commencer, les définitions peuvent inclure les noms d'autres éléments. Dans notre exemple ci-dessus, la première déclaration définit un élément appelé `pets` et peut inclure un seul autre élément, `pet`. De la même façon, l'élément `pet` peut contenir quatre autres éléments : `name`, `age`, `type`, et `color`. Chaque élément est qualifié par le nombre de fois qu'il peut être utilisé dans un document XML. Ce qui se fait grâce à l'astérisque (*), le point d'interrogation (?), et le signe plus (+). Chacun de ces symboles a une signification spécifique :

- astérisque (*) - l'élément peut apparaître zéro ou plusieurs fois
- point d'interrogation (?) - l'élément peut apparaître zéro ou une fois seulement
- signe plus (+) - l'élément doit apparaître au moins une fois

Si un élément n'est qualifié par aucun de ces symboles, alors il peut apparaître une fois et une seule. En conséquence, dans l'exemple ci-dessus, l'élément `pets` étant défini comme contenant l'élément `pet`, et l'élément `pet` étant qualifié par un signe plus, il doit y avoir au moins un élément `pet` dans l'élément `pets`.

PCDATA

#PCDATA est une autre valeur de définition d'élément que vous devez connaître. Cela signifie "parsed character data" (donnée de caractère parsée), et c'est utilisé pour dénoter un contenu qui ne contient que du texte, sans marquage.

Séquences

Enfin, il est tout à fait possible qu'un élément contienne de multiples sous-éléments. Quand ils sont combinés, la liste des éléments est appelée une séquence; on peut les grouper comme suit:

- la virgule (,) est utilisée pour spécifier l'ordre attendu des éléments dans le fichier XML
- les parenthèses (()) sont utilisées pour grouper des éléments
- la barre verticale (|) est utilisée pour indiquer une relation booléenne d'union entre les éléments.

Rassembler tout ça

En examinant la DTD de pets.xml on voit que:

1. L'élément racine du document devrait être pets.
2. L'élément racine, pets, contient au moins un élément pet.
3. Chaque élément pet contient un et un seul élément name, age, type, et color, dans cet ordre.
4. Les éléments name, age, type, et color doivent contenir du texte, sans balises de marquage.

Voici, ci-dessous, une DTD pour la lettre d'un exemple précédent.

```
<!ELEMENT letter (date, addressee, greeting, (paragraph+ | list+)*, closing)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT addressee (name, adresse_un, adresse_deux)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address_one (#PCDATA)>
<!ELEMENT address_two (#PCDATA)>
<!ELEMENT greeting (#PCDATA)>
<!ELEMENT paragraph (#PCDATA | italiques)*>
<!ELEMENT italics (#PCDATA)>
<!ELEMENT list (item+)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT closing (#PCDATA)>
```

Cet exemple est un peu plus compliqué. En le parcourant, nous voyons que:

1. l'élément letter contient un élément date, un élément addressee, un élément greeting, au moins un paragraph ou au moins un élément list, et un élément closing.
2. L'élément date contient du texte brut, sans balises.

3. L'élément destinataire contient un et un seul élément name, adresse_un, et adresse_deux, dans cet ordre.
4. Les éléments name, adresse_un, adresse_deux, et salutation contiennent du texte brut, sans marquage.
5. L'élément paragraphe peut contenir du texte brut, ou l'élément italiques.
6. L'élément italiques contient du texte brut, sans marquage.
7. L'élément liste contient au moins un élément item.
8. Les éléments item et conclusion contiennent du texte brut.

Pour inclure cette DTD dans notre fichier XML, nous devons indiquer où elle se trouve, et puisque cette DTD est dans notre environnement local, et non pas un standard, le lien devrait être inséré dans le document XML, comme suit:

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

<letter>
  <date>
    11 Décembre 2002
  </date>
  <addressee>
    <nom>
      Melville Dewey
    </nom>
    <address_one>
      Columbia University
    </address_one>
    <address_two>
      New York, NY
    </address_two>
  </addressee>
  <greeting>
    Cher Melville,
  </greeting>
  <paragraph>
    J'ai lu vos textes concernant la nature de la bibliothéconomie, et
    <italics>je les trouve tout à fait fascinants</italics>. J'aimerais avoir
    l'occasion de discuter avec vous du rôle du catalogue papier dans les
    bibliothèques aujourd'hui, compte tenu du développement du World Wide Web.
    En particulier, dans quelle mesure des services comme Google ou Amazon.fr affectent-ils
    les demandes de nos lecteurs à l'égard des services de bibliothèque?
    M. Cutter et moi-même discuterons de ces sujets lors de la prochaine conférence annuelle,
    et nous serons disponibles aux dates et heures suivantes:
  </paragraph>
  <list>
    <item>
      Lundi, 2-4
    </item>
    <item>
      Mardi, 3-5
    </item>
    <item>
      Jeudi, 1-3
    </item>
  </list>
  <paragraph>
    Nous espérons que vous pourrez nous rejoindre.
  </paragraph>
  <closing>
    Cordialement, S. R. Ranganathan
  </closing>
</letter>
```

Un processeur XML devrait savoir que l'élément nommé letter est la racine du fichier XML, et que le fichier XML peut être validé en utilisant une DTD locale, non-standardisée, appelée letter.dtd.

Exercice - Ecrire une DTD simple

Dans cet exercice vous examinerez une DTD existante puis vous rédigerez votre propre DTD.

1. Examinez cette DTD qui décrit le contenu du fichier catalog.xml, et répondez aux questions qui suivent:

```
<!ELEMENT catalog      0(caption, structure, work+)>
<!ELEMENT caption      (#PCDATA)>
<!ELEMENT structure    (title, author, type, date)>
<!ELEMENT work          (title, author, type, date)>
<!ELEMENT title         (#PCDATA)>
<!ELEMENT author        (#PCDATA)>
<!ELEMENT type          (#PCDATA)>
<!ELEMENT date          (#PCDATA)>
```

Traduction des éléments : catalog=catalogue, caption=en-tête, structure=structure, work=oeuvre, title=titre, author=auteur, type=genre, date=date.

- A. Combien d'éléments peut contenir l'élément catalog, et lesquels?
 - B. Combien d'oeuvres (work) peut contenir un fichier catalog.xml?
 - C. Un élément title peut-il contenir du contenu balisé? Expliquez.
 - D. Si cette DTD est développée localement, mais qu'elle doit être accessible en dehors du document XML, comment rédigeriez-vous la déclaration de DTD du document XML?
2. Créez une DTD interne pour le fichier ala.xml, et validez le fichier XML en question.
 - A. Ouvrez ala.xml dans le bloc notes.
 - B. Ajoutez une DTD interne au début du fichier, <!DOCTYPE letter []>.
 - C. Entre les crochets ([]), ajoutez la structure d'une déclaration d'élément pour chaque élément qui doit être défini (c'est-à-dire letter, date, address, greeting, etc.). Par exemple, tapez <!ELEMENT para ()> entre les crochets pour l'élément paragraph.
 - D. Définissez le contenu de chaque élément : d'autres noms d'éléments ou #PCDATA, selon la façon dont est structuré le fichier XML. N'oubliez pas d'ajouter ou bien un signe plus (+), un astérisque (*) ou un point d'interrogation (?) pour indiquer le nombre de fois qu'un élément ou une liste d'éléments peuvent apparaître dans le fichier XML.
 - E. Sauvegardez ala.xml.
 - F. Copiez-collez le contenu d'ala.xml dans le bloc-notes.

- G. Ouvrez votre navigateur et validez votre fichier XML en utilisant un formulaire de validation [<http://www.stg.brown.edu/service/xmlvalid/>] .

Part II. Quelques vocabulaires XML spécifiques

Table of Contents

7. XHTML	
Introduction	40
Exercice - Ecrire un document XHTML	43
8. TEI	
Introduction	45
Quelques éléments	45
Exercice	48
9. EAD	
Introduction	49
Exemple	50
10. DocBook	
Introduction	53
Traiter DocBook avec XSLT	55
11. RDF	
Introduction	59
Exercice	61
12. Récolter des métadonnées avec OAI-PMH	
Qu'est-ce que l'Initiative pour les Archives Ouvertes?	62
Le problème	62
La solution	63
Verbes	63
Réponses -- le flux XML	64
Un exemple	67
Conclusion	67

Chapter 7. XHTML



Introduction

XHTML est une implémentation XML de HTML. Dès lors les six règles de la syntaxe XML s'appliquent : il y a un élément racine, les noms des éléments sont sensibles à la casse (bas de casse), les éléments doivent être fermés, ils doivent être correctement emboîtés, les attributs doivent être entre guillemets, et les caractères spéciaux doivent être codés en tant qu'entités. Il existe quelques DTD XHTML, qui vont de DTD très strictes qui interdisent les balises de style et les tableaux, à des DTD plus souples qui, simplement, n'encouragent pas ce type de balises. Les documents XHTML requièrent une déclaration DOCTYPE au début du fichier qui annonce quelle version de XHTML est respectée dans le document. Donc, tout ce qui suit s'applique:

- Votre document doit avoir un et un seul élément html.
- Tous les éléments sont en bas de casse
- Les éléments vides tels que hr et br doivent être fermés comme suit: `<hr />` et `
`
- Les attributs doivent être entre guillemets, comme dans ``
- Vous devez emboîter correctement les éléments
- Les caractères `<`, `>`, et `&` doivent être codés en tant qu'entités

XHTML a quatre attributs "communs" à n'importe quel élément XHTML. Ces attributs sont:

1. id - utilisé pour identifier une position unique dans un fichier
2. title - utilisé pour attribuer une étiquette lisible (par l'homme) à un élément
3. style - qui est utilisé pour des informations du type CSS
4. class - utilisé pour attribuer une étiquette à un élément, en général pour les feuilles de style CSS

En faisant un usage généreux de ces attributs communs, et en leur assignant des valeurs significatives, il est possible de complètement séparer le contenu de la présentation tout en créant des documents accessibles, par toutes sortes de gens tout autant que par des ordinateurs.

Les éléments stylistiques sont découragés dans le but de séparer plus encore le contenu de la présentation. Quand une certaine mise en forme est nécessaire, vous êtes invités à user avec libéralité des feuilles de style. Votre spécification CSS peut résider dans un fichier extérieur, avec un lien dans l'en-tête du document XHTML, ou être directement spécifiée à l'intérieur de chaque élément XHTML grâce à l'attribut style.

Les tableaux font partie de XHTML, et doivent être utilisés pour afficher des données tabulées. Mais l'usage des tableaux pour la mise en page est découragé. Vous devez, plutôt, utiliser les éléments div et span, combinés avec un fichier CSS, pour positionner des blocs de texte sur l'écran.

Vous trouverez ci-dessous un fichier XHTML simple et un fichier CSS représentant une page d'accueil. Le graphisme est pris en charge par le fichier CSS, et même quand le fichier CSS n'est pas utilisé, l'affichage n'est pas si mauvais.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Une page d'accueil simple</title>
  <link rel="stylesheet" href="home.css" type="text/css" />
</head>
<body>
<div class="menu">
  <h3 class="label">Liens</h3>
  <a href="http://infomotions.com/travel/" title="Notes de voyage">Notes de voyage</a><br />
  <a href="http://infomotions.com/musings/" title="Réflexions sur l'information">Réflexions sur l'inf
  <a href="http://infomotions.com/" title="Accueil Infomotions">A propos d'Infomotions</a><br />
</div>
<div class="content">
  <h1>Une page d'accueil simple </h1>
  <p>
    Ceci est une page d'accueil simple qui illustre l'usage de XHTML version 1.0.
  </p>
  <p>
    XHTML n'est pas très différent de HTML.
    XHTML comprend toutes les balises habituelles, telles que la balise pour les références
    hypertextes (liens) et les images. Les tableaux existent toujours dans ce standard, mais n
    à être utilisés pour la mise en page.
  </p>
  <p>
    La transition d'HTML à XHTML est simple tant que vous respectez quelques règles. D'abord,
    assurez-vous de respecter les six règles de la syntaxe XML. Ensuite, méfiez-vous des balises
    de style (éléments) tels que bold, italics, et en particulier font. Enfin, utilisez largemen
    div et span. Utilisés avec les fichiers CSS, ils vous permettront de positionner et de forma
    facilement des blocs de texte à l'écran.
  </p>
  <hr />
  <p class="footer">
    Auteur: Eric Lease Morgan &lt;<a
    href="mailto:eric_morgan@infomotions.com">eric_morgan@
    infomotions.com</a>&gt;
    <br />
    Date: 19-01-2003
    <br />
    URL: <a href="./home.html">./home.html</a>
  </p>
</div>
</body>
</html>
```



```
h1, h2, h3, h4, h5, h6 {
  font-family: helvetica, sans-serif;
}

p {
  font-family: times, serif;
  font-size: large;
}

p.footer {
  font-size: small;
}

div.menu {
  position: absolute;
  margin-right: 82%;
  text-align: right;
  font-size: small;
}

div.content {
  position: absolute;
  margin-left: 22%;
  margin-right: 3%;
}

a:hover {
  color: red;
  background-color: yellow;
}
```

Si vous affichez ces fichiers dans un navigateur compatible CSS, cela devrait donner quelque chose comme ça:



Exercice - Ecrire un document XHTML

Dans cet exercice, vous apprendrez à créer un document en utilisant une DTD standard, en l'espèce XHTML.

1. Baliser le fichier ala.txt en tant que document XHTML.
 - A. Ouvrez ala.txt dans le bloc-notes.
 - B. Sauvegardez le fichier sous le nom ala.html.
 - C. Ajoutez au début du fichier la déclaration XML, `<?xml version="1.0"?>`.
 - D. Ajoutez la déclaration de type de document (DTD), `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
 - E. Ajoutez les éléments ouvrant et fermant d'html (i.e. `<html>`).
 - F. Ajoutez les sections head et body, ouvrantes et fermantes (i.e. `<head>`, `<body>`).
 - G. Ajoutez un élément title à la section head (i.e. `<title>`).
 - H. Ajoutez un lien qui pointe vers l'endroit où se trouvera votre fichier CSS (i.e. `<link rel="stylesheet" href="ala.css" type="text/css" />`).
 - I. En utilisant seulement les éléments p et br, balisez le corps de la lettre en vous assurant de bien ouvrir et refermer chaque élément (i.e. `<p>`, `
`).
 - J. Sauvegardez à nouveau votre fichier, et affichez le dans votre navigateur web.

- K. Créez un nouveau document dans le bloc-notes, et sauvegardez le sous le nom ala.css.
- L. Ajoutez seulement deux sélecteurs au fichier CSS, un pour chaque implémentation différente de l'élément p (i.e. `p { font-family: times, serif; font-size: large }` et `p.closing { text-align: right }`).
- M. Ajouter l'attribut commun "class" au dernier paragraphe de votre lettre en lui donnant la valeur conclusion (i.e. `<p class="closing">`).
- N. Sauvegardez à nouveau votre fichier, et affichez le dans votre navigateur web.

Chapter 8. TEI



Introduction

TEI, ou Text Encoding Initiative, est un grand-père des langages de marquage. TEI, qui a commencé comme SGML et n'est devenu XML que relativement récemment, est utilisé le plus souvent dans la communauté des sciences humaines pour baliser des oeuvres littéraires (prose et poèmes). Souvent dans ces communautés on scanne des documents originaux, qu'on convertit en texte en utilisant des techniques de reconnaissance optique de caractères (OCR), on corrige les erreurs, et on balise le résultat en TEI. Dans l'idéal, on dispose d'un original du livre ou du manuscrit à côté de la version digitale en TEI. Avec les deux, un spécialiste peut analyser le texte très finement et ainsi créer un nouveau savoir.

La DTD TEI est très riche et "bavarde". Elle contient des éléments qui couvrent toutes les figures littéraires (paragraphe, strophes, chapitres, notes de bas de page, etc.). Puisque les documents TEI cherchent, la plupart du temps, à refléter le plus fidèlement possible les documents originaux, la DTD contient des balises qui peuvent indiquer des choses comme les sauts de page ou les numéros de ligne dans le texte original. Il y a des balises pour les références croisées et les liens hypertextes. Il y a même des balises pour le commentaire éditorial, l'interprétation, et l'analyse. La DTD est si prolifique que certains experts de TEI suggèrent de ne l'utiliser qu'en partie. En pratique, de nombreuses institutions qui utilisent la DTD TEI utilisent ce qu'on appelle couramment TEILite, une version simplifiée de la DTD qui contient les définitions des éléments les plus couramment utilisés par la plupart des gens.

Quelques éléments

Ce texte n'a pas vocation à donner plus qu'une très sommaire introduction à TEI. Cette section décrit les éléments TEI de base, et la façon dont les documents TEI peuvent être traités par les outils XML.

Le plus simple des documents TEI contient des sections en-tête (teiHeader) et texte (text). La section teiHeader contient un certain nombre de sous-éléments qui fournissent des métadonnées à propos du document. La section text est divisée en trois sous-sections (front, body, et back). Voici une liste des principaux éléments TEI avec une brève description pour chaque:

- TEI.2 - l'élément racine du document TEILite
- teiHeader - qui contiendra les métadonnées du document TEI

- fileDesc - un sous-élément obligatoire de teiHeader qui décrit le fichier TEI
- titleStmt - où apparaîtront l'auteur et le titre de l'oeuvre
- title - le titre du document
- author - l'auteur du document
- publicationStmt - texte libre décrivant la façon dont est publié le document
- sourceDesc - une déclaration décrivant d'où est tiré le texte
- text - c'est l'élément dans lequel le texte du document est stocké
- front - dénote la couverture d'un livre ou d'un manuscrit
- body - le coeur du sujet, le livre ou le manuscrit lui-même
- back - le quatrième de couverture du livre ou du manuscrit
- date - une date
- p - un paragraphe
- lg - un groupe de ligne, par exemple dans un poème
- l - une ligne dans un groupe de lignes

En utilisant les éléments ci-dessus il est possible de créer un document TEI valide, quoiqu'un peu limité, comme celui-ci:

```
<TEI.2>

<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Getting Started with XML</title>
      <author>Eric Lease Morgan</author>
    </titleStmt>
    <publicationStmt><p>Originally published in <date>march 2003</date> for an Infopeople workshop.</p>
    <sourceDesc><p>There was no original source; this document was born digital.</p></sourceDesc>
  </fileDesc>
</teiHeader>

<text>

  <front></front>

  <body>

    <p>
      Getting started with XML is a workshop and manual providing an overview of XML and how it can be
    </p>

    <lg>
      <l>There lives a young girl in Lancaster town,</l>
      <l>Whose hair is a shiny pale green.</l>
      <l>She lives at the bottom of Morgan's farm pond,</l>
      <l>So she's really too hard to be seen.</l>
    </lg>

  </body>
```

```
<back></back>

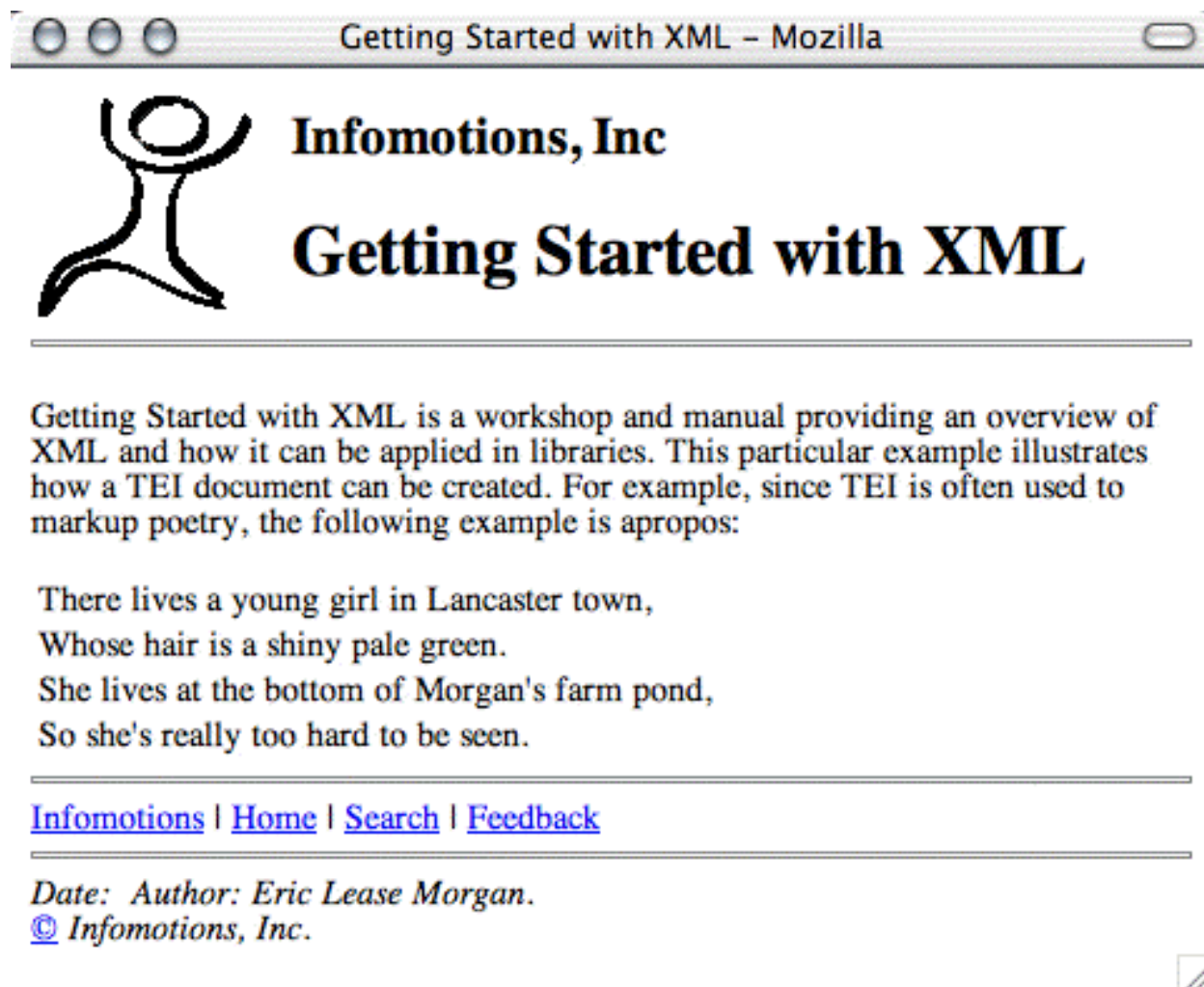
</text>

</TEI.2>
```

Les fichiers TEI ont historiquement été créés puis indexés/restitués grâce à un ensemble de programmes intermédiaires tels que XPAT. La disponibilité croissante de technologies XML telles que CSS et XSLT ont fait qu'un certain nombre de feuilles de style sont désormais disponibles. Le site web officiel de la TEI propose des liens vers ces ressources, et les feuilles de style XSLT fonctionnent assez bien.

Si vous sauvegardez le texte TEI ci-dessus sous le nom `tei-getting.xml`, vous pourriez créer un très beau document XHTML en utilisant les feuilles de style XSLT et `xsltproc` comme ceci: **`xsltproc tei-stylesheets/teihtml.xsl tei-getting.xml`**.

Voici le résultat:



Attention. La feuille de style XSL est configurée de telle façon qu'elle sauvegarde toujours les documents sous le nom `index.html`, même si vous spécifiez l'option `-o`. Vous devrez éditer le fichier nommé `teixsl-html/teihtml-param.xml` pour mettre le résultat XHTML à votre sauce.

Exercice

Dans cet exercice vous afficherez un fichier TEI en utilisant CSS et XSLT.

1. Afficher un fichier TEI en utilisant CSS

- A. Ouvrez le fichier `tei.xml` dans le bloc-notes.
- B. Ajoutez au début du fichier l'instruction XML suivante: `<?xml-stylesheet type='text/css' href='tei-dancer.css'>`
- C. Sauvegardez le fichier.
- D. Ouvrez `tei.xml` dans votre navigateur web. Appréciez le résultat.
- E. Changez la valeur de la balise `href` dans l'instruction XML de `tei.xml` et remplacez la par `tei-oucs.css`.
- F. Sauvegardez le fichier.
- G. Ouvrez `tei.xml` dans votre navigateur web. A nouveau : appréciez le résultat.

2. Transformez le fichier TEI en XHTML

- A. Exécutez la commande suivante : **`xsltproc teixsl-html/teihtml.xsl tei.xml`**
- B. Ouvrez le fichier `index.html` qui en résulte dans votre navigateur web. Intéressant, non?
- C. Editez le fichier `tei-stylesheets/teihtml-param.xsl` et changez la valeur de institution : remplacez Oxford University Computing Services par votre nom.
- D. Sauvegardez `teixsl-html/teihtml-param.xsl`.
- E. Répétez les étapes #A et #B. Super!

Chapter 9. EAD



Introduction

EAD signifie Encoded Archival Description (Description Archivistique Encodée), et il s'agit d'un vocabulaire SGML/XML utilisé pour baliser des outils d'aide à la recherche archivistique. Comme TEI, EAD est profondément enraciné dans SGML, et comme TEI, EAD n'est devenu compatible XML que relativement récemment.

Comme vous le savez, ou pas, les outils d'aide à la recherche archivistique sont des descriptions formelles de pièces d'archives institutionnelles. Cela ne se limite pas aux manuscrits, aux notes, aux lettres, aux œuvres publiées ou non d'individus ou de groupes : cela comprend aussi, désormais, des programmes et des données informatiques, des films, des vidéos, des enregistrements sonores, des objets. Du fait du nombre de pièces individuelles dans ces archives, les items ne sont généralement pas décrits individuellement mais en tant que collections. Qui plus est, les items ne sont généralement pas organisés par sujet mais plus probablement par date dans la mesure où l'ordre chronologique de création des items incarne le développement des idées représentées par la collection. Du fait de ces caractéristiques, les pièces d'archive ne sont habituellement pas décrites avec un format MARC mais, de plus en plus, grâce à EAD.

Selon la DTD EAD, seuls quelques éléments sont nécessaires pour créer un document EAD valide, mais ce type de document ne sera pas une très bonne aide à la recherche. En conséquence, le guide EAD Application Guidelines suggère les éléments suivants:

- ead - la racine du document EAD
- eadheader - cet élément encadrera les métadonnées relatives à un document EAD
- eadid - le code unique d'un document EAD
- filedesc - cet élément encadrera la description bibliographique du document EAD
- titlestmt - ici se trouveront des choses telles que l'auteur et le titre d'un document EAD
- titleproper - le titre du document EAD
- author - les noms des individus ou groupes qui ont créé le document EAD
- publicationstmt - concerne les informations de publication

- publisher - le nom du distributeur du document EAD
- date - une date
- profiledecs - ici se trouveront un ensemble d'informations à propos de la procédure d'encodage d'EAD
- creation - ici pourront être placées des noms de personnes ou de lieux en liaison avec la procédure d'encodage EAD
- language - une liste des langues présentes dans le document EAD
- langusage - un élément dénotant une langue spécifique
- archdesc - contient l'essentiel de l'aide archivistique; c'est ici que la pièce d'archive est décrite.
- did - un élément pour décrire une unité individuelle
- repository - l'institution ou l'agence responsable du groupe d'items de la collection
- corpname - un nom de collectivité
- origination - le nom de la personne ou du groupe responsable du rassemblement des pièces de la collection
- persname - un nom de personne
- famname - un nom de famille
- unittitle - un titre pour le matériel faisant l'objet de la description
- unitdate - année, mois et jour de création du matériel faisant l'objet de la description
- physdesc - un élément qui permet la description des caractéristiques physiques de la collection
- unitid - un numéro de référence unique -- numéro de contrôle -- pour le matériel faisant l'objet de la description
- abstract - un résumé décrivant la collection
- bioghist - une brève histoire ou une chronologie plaçant l'archive dans son contexte
- scopecontent - un résumé décrivant l'éventail des sujets traités par l'archive
- controlaccess - un élément utilisé pour dénoter des termes de vocabulaires normalisés dans d'autres collections ou d'autres index.
- dsc - un élément regroupant des groupes hiérarchisés de sous-ensembles de la collection
- c - cet élément décrit une partie logique du matériel décrits
- container - habituellement un nombre entier utilisé en conjonction avec certains attributs pour signaler l'étendue physique des pièces d'archives décrites

Ouf!

Exemple

Voici un fichier EAD. Pouvez-vous trouver de quoi il s'agit?

```

<ead>
  <eadheader>
    <eadid>
      ELM001
    </eadid>
    <filedesc>
      <titlestmt>
        <titleproper>
          Collection Eric Lease Morgan
        </titleproper>
        <author>
          Créée par Eric Lease Morgan
        </author>
      </titlestmt>
      <publicationstmt>
        <publisher>
          Infomotions, Inc.
        </publisher>
        <date>
          20030218
        </date>
      </publicationstmt>
      <profiledesc>
        <creation>
          Ce fichier a été créé avec un éditeur de texte simple.
        </creation>
        <langusage>
          Ce fichier ne contient qu'une langue, le
          <language>
            Français
          </language>
        </langusage>
      </profiledesc>
    </filedesc>
  </eadheader>
  <archdesc level='otherlevel'>
    <did>
      <repository>
        <corpname>
          Infomotions, Inc.
        </corpname>
      </repository>
      <origination>
        <persname>
          Eric Lease Morgan
        </persname>
      </origination>
      <unittitle>
        Documents
      </unittitle>
      <unitdate>
        1980-2001
      </unitdate>
      <physdesc>
        Un ensemble de quatre boîtes contenant pour l'essentiel des papiers A4, conservées dans mon gara
      </physdesc>
      <unitid>
        B0001
      </unitid>
      <abstract>
        Au fur et à mesure des années, j'ai conservé diverses choses que j'avais écrits.
        Cette collection regroupe l'essentiel de ces papiers. Je suis bien persuadé qu'elle contribuera à
        l'avancée de la connaissance après ma mort.
      </abstract>
    </did>
    <biohist>
      <p>
        Eric est né à Lancaster, PA. Il a fait ses études à Bethany, WV.
        Il a vécu à Charlotte et Raleigh, NC pendant quinze ans.
      </p>
    </biohist>
  </archdesc>

```

Il vit désormais à South Bend, IN.

</p>
</biohist>

<scopecontent>

<p>

Cette collection consiste en travaux prépubliés, photographes, dessins, et de courriels importants conservés depuis des années.

</p>

</scopecontent>

<controlaccess>

<p>

Il est peu probable qu'on trouve un vocabulaire standardisé dans d'autres systèmes qui conserveraient un matériel similaire.

</p>

</controlaccess>

<dsc type='othertype'>

<c level='otherlevel'>

<did>

<container type='box'>

1

</container>

<unittitle>

Boîte 1

</unittitle>

<unitdate>

1980-1984

</unitdate>

</did>

<did>

<container type='box'>

1

</container>

<unittitle>

Boîte 2

</unittitle>

<unitdate>

1985-1995

</unitdate>

</did>

<did>

<container type='box'>

1

</container>

<unittitle>

Boîte 3

</unittitle>

<unitdate>

1995-1998

</unitdate>

</did>

<did>

<container type='box'>

1

</container>

<unittitle>

Boîte 4

</unittitle>

<unitdate>

1999-2001

</unitdate>

</did>

</c>

</dsc>

</archdesc>

</ead>

Chapter 10. DocBook



Introduction

DTD est une DTD utilisée pour baliser des documents informatiques. La DTD a évolué depuis sa création il y a plus de dix ans par l'éditeur O'Reilly, qui publie de nombreux ouvrages informatiques. Comme XHTML et TEI, DocBook sert à baliser des textes narratifs, mais DocBook dispose aussi d'un certain nombre d'éléments spécifiques au thème de l'informatique. Parmi ces éléments, il y a des choses comme les copies d'écran (screenshot), les programmes (programlisting), et les commandes (command).

Il y a un large éventail de types de documents de base dans DocBook. Les plus courants sont book et article. Les documents book peuvent contenir des choses comme une préface, des chapitres, et des annexes. Qui peuvent être sub-divisées en sections contenant des paragraphes, des listes, des figures, des exemples, et des morceaux de programmes. (Ce manuel/atelier est balisé avec la DTD DocBook.) Les articles sont comme les livres (book), mais ne contiennent pas de chapitres.

Pour créer un fichier DocBook valide, il faut que le fichier contienne une déclaration de type de document (DTD) qui identifie la version de DocBook utilisée. La DTD DocBook évoluant, il y a des déclarations différentes. Voici une déclaration pour un article utilisant la version 4.2 de la DTD:

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
```

Voici une liste de quelques uns des éléments les plus couramment utilisés dans un article DocBook:

- article - l'élément racine
- articleinfo - cet élément balise les métadonnées de l'article
- author - cet élément contient les informations concernant le créateur de l'article
- firstname - le prénom d'un auteur
- surname - le nom d'un auteur

- email - une adresse email
- pubdate - la date de publication de l'article
- abstract - une description de l'article
- title - cet élément est souvent utilisé dans l'ensemble du document book ou article, pour donner un en-tête à des éléments tels que article, book, section, example, figure, etc.
- section - une partie générique d'un livre (book) ou d'un article
- para - un paragraphe
- command - cet élément est utilisé pour indiquer une commande informatique, habituellement telle qu'elle est entrée en ligne de commande
- figure - généralement utilisé pour une image
- graphic - l'endroit prévu pour une image
- ulink - une référence de lien hypertexte
- programlisting - tout ou partie d'un programme informatique
- orderedlist - une liste à lettres
- itemizedlist - une liste à puces
- listitem - un item dans une liste, à lettres ou à puces

En utilisant certaines des balises ci-dessus, on peut créer l'article DocBook suivant.

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
"http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<article>
  <articleinfo>
    <author>
      <firstname>
        Eric
      </firstname>
      <surname>
        Morgan
      </surname>
    </author>
    <title>MyLibrary: un site web en base de données pour les bibliothèques</title>
    <pubdate>
      Février 2003
    </pubdate>
    <abstract>
      <para>
        Cet article décrit un logiciel de gestion de site web en base de données appelé MyLibrary.
      </para>
    </abstract>
  </articleinfo>
  <section>
    <title>Introduction</title>
    <para>
      Cet article décrit un logiciel de gestion de site web en base de données appelé MyLibrary.
    </para>
  </section>
  <section>
    <title>Plus qu'une belle gueule</title>
```

```
<para>
  MyLibrary est dérivé d'une interface personnalisable pour l'accès aux ressources de la bibli
  A mesure que l'application murissait, c'est devenu un système pour créer et gérer un site w
bibliothèque en utilisant l'idée d'architecture de l'information. Spécifiquement, le système MyLibra
donne à une bibliothèque les moyens de créer des choses comme:
</para>
<itemizedlist>
  <listitem>
    <para>
      une carte du site
    </para>
  </listitem>
  <listitem>
    <para>
      un index du site
    </para>
  </listitem>
  <listitem>
    <para>
      des vocabulaires contrôlés
    </para>
  </listitem>
  <listitem>
    <para>
      un outil de recherche puissant
    </para>
  </listitem>
  <listitem>
    <para>
      des listes de ressources qu'on peut feuilleter
    </para>
  </listitem>
  <listitem>
    <para>
      des interfaces personnalisables, optionnelles
    </para>
  </listitem>
</itemizedlist>
<para>
  En ligne de commande, vous pouvez vérifier que votre installation MyLibrary fonctionne correct
en tapant la commande suivante:
  <command>
    ./mylibrary.pl
  </command>
  . Vous devriez obtenir un résultat en HTML, à lire dans un navigateur web.
</para>
<para>
  Pour plus d'informations sur MyLibrary, voir:
  <ulink url="http://dewey.library.nd.edu/mylibrary/">
    http://dewey.library.nd.edu/mylibrary/
  </ulink>
  .
</para>
</section>
</article>
```

Traiter DocBook avec XSLT

DocBook bénéficie, c'est un de ses avantages, d'un bon support. Plusieurs feuilles de style XSLT sont disponibles pour DocBook : elles vous permettent d'utiliser votre processeur XSLT préféré pour transformer des fichiers DocBook en fichiers XHTML, HTML, PDF, en pages man Unix, en fichiers d'aide Windows, ou même en diapositives du type PowerPoint. Par exemple, vous pouvez utiliser une commande comme celle-ci pour transformer un fichier DocBook en HTML: **xsltproc -o docbook-article.html docbook-stylesheets/html/docbook.xsl docbook-article.xml** . Vous obtenez un fichier HTML nommé docbook-article.html qui, dans un navigateur web, ressemble à ça.



MyLibrary: A Database-Driven Website System for Libraries

Eric Morgan

February, 2003

Abstract

This article describes a database-driven website application for libraries called MyLibrary.

Table of Contents

[Introduction](#)

[More than a pretty face](#)

Introduction

This article describes a database-driven website application for libraries called MyLibrary.

More than a pretty face

MyLibrary has its roots in a customizable interface to collections of library information resources. As the application has matured, it has become a system for creating and managing a library's website by manifesting the ideas of information architecture. Specifically, the MyLibrary system provides the means for a library to create things like:

- top down site maps
- bottom-up site indexes
- controlled vocabularies
- a means for power searching
- browsable lists of resources
- optional, user-specified customizable interfaces

From the commandline you see if your MyLibrary installation is working correctly by issuing the following command: `./mylibrary.pl`. The result should be a stream of HTML intended for a Web browser.

For more information about MyLibrary, see: <http://dewey.library.nd.edu/mylibrary/>.



Vous pouvez aussi créer des documents PDF à partir de fichiers DocBook en utilisant quelque chose comme le programme FOP et une feuille de style appropriée. Par exemple, cette commande crée un document PDF: `fop.sh -xml docbook-article.xml -xsl /docbook/stylesheets/fo/docbook.xsl docbook-article.pdf`. Vous obtenez un document PDF qui ressemble à ça:

MyLibrary: A Database-Driven Website System for Libraries

Eric Morgan

February, 2003

This article describes a database-driven website application for libraries called MyLibrary.

Table of Contents

Introduction	1
More than a pretty face	1

Introduction

This article describes a database-driven website application for libraries called MyLibrary.

More than a pretty face

MyLibrary has its roots in a customizable interface to collections of library information resources. As the application has matured, it has become a system for creating and managing a library's website by manifesting the ideas of information architecture. Specifically, the MyLibrary system provides the means for a library to create things like:

- top down site maps
- bottom-up site indexes
- controlled vocabularies
- a means for power searching
- browsable lists of resources
- optional, user-specified customizable interfaces

From the commandline you see if your MyLibrary installation is working correctly by issuing the following command: `/mylibrary.pl`. The result should be a stream of HTML intended for a Web browser.

For more information about MyLibrary, see: <http://dewey.library.nd.edu/mylibrary/> [<http://dewey.library.nd.edu/mylibrary/>].

voudriez, vous pouvez toujours créer vos feuilles de style XSLT en utilisant celles fournies par la communauté DocBook comme canevas de départ (template).

Chapter 11. RDF



Introduction

Resource Description Framework (RDF) est une proposition permettant d'encoder de façon consistante des métadonnées dans une syntaxe XML. L'idée grandiose derrière RDF est la création du web sémantique. Si tout le monde créait une description RDF de ses documents, alors les ordinateurs seraient capables de trouver des relations entre les documents que les humains ne découvriraient pas nécessairement. A première vue, la syntaxe semble un peu écrasante, mais en fait ça n'est pas si difficile. Juré.

RDF ressemble beaucoup à l'idée d'encoder des métadonnées dans les balises meta des documents HTML. En HTML, les balises meta définissent d'abord le nom de la balise meta, disons, title. Ensuite, l'attribut content de la balise meta HTML donne la valeur du titre, par exemple Autant en emporte le vent. Ainsi la balise meta suivante pourrait apparaître dans un document HTML: `<meta name="title" content="Autant en emporte le vent"/>`. Ces paires nom/valeur doivent décrire le document HTML dans lequel elles se trouvent. Les documents HTML ont des URLs. Ces trois choses (nom, valeur, url) forment ce qu'on appelle, dans le langage RDF, un triplet. Dans RDF tout tient à la création de ces triplets, à la création de paires nom/valeur et à leur usage pour décrire le contenu des URLs.

Il n'est pas rare d'utiliser le Dublin Core pour créer des paires nom/valeur dans les fichiers RDF. Le Dublin Core fournit un ensemble réellement standard de noms d'éléments utilisés pour décrire des ressources Internet. Les quinze noms d'éléments de base sont:

1. title
2. creator
3. subject
4. description
5. publisher
6. contributor
7. date
8. type

9. format
10. identifier
11. source
12. language
13. relation
14. coverage
15. rights

En incorporant dans votre document XML non seulement la DTD RDF mais aussi les balises ("name spaces") pour RDF et Dublin Core, vous pouvez décrire le contenu qui se trouve à d'autres URLs de façon standardisée. Le fichier RDF valide ci-dessous décrit trois sites web en utilisant comme paires nom/valeur quelques éléments Dublin Core. Une fois passé la DTD et les définitions de "name spaces", la seule partie délicate du fichier est l'élément `rdf:Bag`. Cet élément sert à inclure des listes d'items similaires. Ici, une liste de mots sujets.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF PUBLIC "-//DUBLIN CORE//DCMES DTD 2002/07/31//EN"
"http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-xml-dtd.dtd">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.AcronymFinder.com/">
    <dc:title>Acronym Finder</dc:title>
    <dc:description>Acronym Finder est une base de données interrogeable par le world wide
web (WWW), qui contient plus de 169.000 abréviations et acronymes concernant l'informatique,
la technologie, les télécommunications, ainsi que des acronymes et abréviations militaires.</dc:d
    <dc:subject>
      <rdf:Bag>
        <rdf:li>Astronomie</rdf:li>
        <rdf:li>Littérature</rdf:li>
        <rdf:li>Mathématiques</rdf:li>
        <rdf:li>Musique</rdf:li>
        <rdf:li>Philosophie</rdf:li>
      </rdf:Bag>
    </dc:subject>
  </rdf:Description>

  <rdf:Description rdf:about="http://dewey.library.nd.edu/eresources/astronomy.html">
    <dc:title>All Astronomy resources</dc:title>
    <dc:description>Il s'agit d'une liste de toutes les ressources concernant l'astronomie
présentes sur ce site.</dc:description>
    <dc:subject>
      <rdf:Bag>
        <rdf:li>Astronomie</rdf:li>
        <rdf:li>Mathématiques</rdf:li>
      </rdf:Bag>
    </dc:subject>
  </rdf:Description>

  <rdf:Description rdf:about="http://dewey.library.nd.edu/eresources/literature.html">
    <dc:title>All Literature resources</dc:title>
    <dc:description>Il s'agit d'une liste de toutes les ressources concernant la littérature
présentes sur ce site.</dc:description>
    <dc:subject>
      <rdf:Bag>
        <rdf:li>Littérature</rdf:li>
        <rdf:li>Philosophie</rdf:li>
      </rdf:Bag>
    </dc:subject>
```

```
</rdf:Description>
</rdf:RDF>
```

Les codes RDF peuvent être insérés dans d'autres fichiers XML ou constituer des documents autonomes. Par exemple, une partie de code en RDF peut être insérée dans l'élément `notesStmt` de fichiers TEI et fournir un moyen standardisé de décrire le contenu des fichiers. Des ensembles de fichiers TEI pourraient être lus par des ordinateurs et, par là même, un catalogue serait constitué. Malheureusement, cela ne marche pas toujours parce que la DTD TEI, par exemple, ne supporte pas l'addition de données RDF. Même si le fichier TEI est syntactiquement valide, la sémantique décrite par la DTD ne prend pas RDF en compte. Pour inclure des données RDF dans des documents HTML/XHTML, le standard suggère d'inclure dans la balise header du fichier HTML un élément lien qui pointe vers la description RDF, quelque chose comme ça : `<link rel="meta" type="application/rdf+xml" href="monfichier.rdf" />`. Un logiciel peut alors suivre l'attribut href (le lien) pour pouvoir lire le fichier RDF.

Exercice

1. Dans cet exercice vous examinerez les caractéristiques des documents Internet qui utilisent RDF.
 - A. Ouvrez le fichier `rdf.xml` dans le bloc-notes.
 - B. Sélectionnez la totalité du contenu du fichier et copiez le.
 - C. Avec le navigateur web, allez à l'adresse du RDF Validator [<http://www.w3.org/RDF/Validator/>].
 - D. Collez le texte copié dans la zone prévue du formulaire et envoyez.
 - E. Examinez le graphique résultat, qui devrait ressembler à quelque chose comme ç :



- F. En utilisant `rdf.xml` comme modèle, effacez tous les `rdf:Descriptions` sauf un, et modifiez celui qui reste en décrivant la page d'accueil du site web de votre bibliothèque.
- G. Renouvelez les opérations #B à #E.

Chapter 12. Récolter des métadonnées avec OAI-PMH



Note: Ceci est une édition d'un article publié par ailleurs, Eric Lease Morgan "What is the Open Archives Initiative?" *interChange: Newsletter of the International SGML/XML User's Group* 8(2):June 2002, pgs. 18-22.

Cet article décrit les objectifs de l'Initiative pour les Archives Ouvertes (Open Archives Initiative - OAI) et illustre une façon d'implémenter la version 1.1 du protocole. A l'heure qu'il est, le protocole a été renommé Initiative Archives Ouvertes-Protocole pour la Récupération de Métadonnées (Open Archives Initiative-Protocol for Metadata Harvesting, ou OAI-PMH), qui en est maintenant à sa version 2.0. Mais cela ne devrait pas vous dissuader de lire ce chapitre, dont la grande majorité des informations reste valide.

Qu'est-ce que l'Initiative pour les Archives Ouvertes?

En un phrase, l'Initiative pour les Archives Ouvertes (OAI) est un protocole qui s'ajoute à HTTP pour distribuer, rassembler, et fédérer des métadonnées. Le protocole est exprimé en XML. Ce chapitre décrit les problèmes qu'OAI tente de résoudre et la façon dont il est censé marcher. A la fin de ce chapitre, vous en saurez un peu plus à propos de l'OAI et peut-être voudrez-vous créer votre propre répertoire OAI ou même devenir fournisseur de service. La page ressource sur l'OAI est <http://www.openarchives.org/> [<http://www.openarchives.org/>].

Le problème

Très sommairement, le problème est : "Comment identifier et localiser l'information dont j'ai besoin?"

Il semble que nous buvions tous à la même source et que nous souffrions, au minimum, d'une légère surcharge d'information. Si vous utilisez les moteurs de recherche Internet pour trouver l'information dont vous avez besoin, vous récupérez littéralement des milliers de résultats. Assez souvent, les résultats affichés sont décrits de façon inadéquate, et cliquer sur un lien particulier est un peu un pari. Les bases de données bibliographiques --des index de la littérature universitaire publiée de façon formelle-- déboussolent l'utilisateur par un nombre trop important d'options de recherche et proposent rarement le plein texte des articles identifiés. Ces bases de données se contentent de fournir à l'utilisateur une référence qu'il lui faudra emporter jusqu'à la bibliothèque en espérant que l'article est quelque part sur les rayons.

Du point de vue du fournisseur de contenu, le problème peut être décrit à l'inverse : "Comment puis-je faire connaître aux gens les données et l'information que je diffuse?"

De nombreuses personnes (les fournisseurs de contenus) ont des informations à partager avec d'autres personnes qui en ont réellement besoin. Rassembler, organiser, et maintenir l'information n'est qu'une partie du travail. Sans accès ces efforts sont vains. En plus de tout ça, il peut y avoir des groupes de fournisseurs de contenus qui ont des informations communes, par exemple des sujets communs (littérature, mathématiques, jardinage), des formats de fichiers (images, textes, sons), ou une communauté (une bibliothèque, une entreprise, un groupe d'utilisateur). Ces ensembles de personnes peuvent vouloir coopérer en fusionnant leurs contenus dans un moteur de recherche commun, économisant ainsi le temps de l'utilisateur en réduisant le nombre de bases de données que les gens doivent interroger et en donnant accès au contenu lui-même.

La solution

OAI tente de résoudre ces problèmes en développant une méthode (un protocole qui s'ajoute à HTTP) pour partager les métadonnées enregistrées dans des bases de données accessibles par Internet. Le protocole définit deux entités et aussi le langage grâce auquel elles communiquent. La première entité est appelée "fournisseur de contenus" (data provider) ou "répertoire" (repository). Par exemple, un fournisseur de contenus possède une collection d'images numériques. Chacune de ces images peut être décrite par un ensemble de qualités : titre, numéro d'accès, données, résolution, description narrative, etc. Ou bien un fournisseur de contenus peut proposer une archive de pre-prints -- une collection d'articles en pré-publication, et ainsi chaque article de l'archive peut être décrit grâce des données auteur, données, résumé, et peut-être sujet. Autre exemple : une liste de personnes, expertes dans un champ d'étude donné. Les qualités décrivant cette collection peuvent être le nom, l'adresse e-mail, l'adresse postale, le numéro de téléphone, l'affiliation institutionnelle.

Donc le but de la première entité OAI --le fournisseur de contenus-- est de faire connaître les qualités de sa collection --ses métadonnées-- à une seconde entité, un "fournisseur de service". Le but du fournisseur de service est de récolter les métadonnées d'un ou plusieurs fournisseurs de contenus pour créer un service avec une certaine valeur ajoutée. Ce service n'est pas défini par le protocole, mais peut consister par exemple en un annuaire imprimé, un index fédéré avec possibilité de recherche, un site miroir pour un fournisseur de contenus, un service d'alerte, un fil de news, etc.

En résumé, OAI définit deux entités (fournisseur de contenus et fournisseur de service) et un protocole qui permet à ces deux entités de partager des métadonnées. Dans la suite de ce chapitre, nous allons décrire le protocole un peu plus en détail.

Verbes

Le protocole OAI consiste seulement en quelques "verbes" (pensez "commandes"), et un ensemble de réponses XML standardisées. Tous les verbes sont transmis par le fournisseur de service au fournisseur de contenus par une requête HTTP. Il s'agit d'un ensemble d'une ou plusieurs paires nom/valeur inscrites dans une URL (comme dans la méthode GET) ou encodées dans l'en-tête HTTP (comme dans la méthode POST). La plupart des verbes peuvent être précisés par des paires nom/valeur. Le verbe le plus simple est "Identify", et voici un exemple de la façon dont on peut transmettre une requête à un fournisseur de contenus grâce à une commande GET :

`http://www.infomotions.com/alex/oai/?verb=Identify` [`http://www.infomotions.com/alex/oai/?verb=Identify`]

L'exemple ci-dessus présume qu'il y a une application capable de traiter OAI dans le répertoire /alex/oai de l'hôte `www.infomotions.com`. Cette application récupère la paire nom/valeur, `verb=Identify`, et renvoie un flux XML qui confirme qu'elle est bien un fournisseur de contenus OAI.

Les autres verbes fonctionnent de façon similaire, mais peuvent utiliser plusieurs qualifications par des paires nom/valeur supplémentaires. Par exemple, la requête web suivante demande une notice, au format de métadonnées Dublin Core, décrivant l'ouvrage de Mark Twain *The Prince And The Pauper*:

`http://www.infomotions.com/alex/oai/?verb=GetRecord&metadataPrefix=oai_dc&identifier=twain-prince-30`
[`http://www.infomotions.com/alex/oai/?verb=GetRecord&metadataPrefix=oai_dc&identifier=twain-prince-30`]

A nouveau, l'application placée dans le répertoire /alex/oai récupère la requête GET et renvoie un flux XML en

réponse.

Les six verbes du protocole sont énumérés et brièvement décrits ci-dessous:

1. Identify - Ce verbe est utilisé pour vérifier qu'un service particulier est bien un répertoire OAI. La réponse à une requête Identify comprend le nom du service, l'URL à laquelle le service peut être consulté, le numéro de version du protocole géré par le fournisseur, et l'adresse e-mail à laquelle on peut s'adresser pour un complément d'information. C'est de loin le verbe le plus simple. Exemple:

<http://www.infomotions.com/alex/oai/?verb=Identify> [<http://www.infomotions.com/alex/oai/?verb=Identify>]

2. ListMetadataFormats - Les métadonnées prennent bien des formes, et cette commande interroge le fournisseur pour obtenir une liste des formats gérés. Pour être valide OAI, un fournisseur doit au moins être capable de gérer le Dublin Core. (Pour plus d'informations à propos du format de métadonnées Dublin Core, voir <http://dublin.or/> et <http://www.iso.or/standards/resources/Z39-85.pdf>.) Exemple:

<http://www.infomotions.com/alex/oai/?verb=ListMetadataFormats>
[<http://www.infomotions.com/alex/oai/?verb=ListMetadataFormats>]

3. List sets - Les données présentes dans un répertoire ne sont pas nécessairement homogènes : on peut trouver des informations à propos de plus d'un sujet, et dans plus d'un format. En conséquence le verbe List sets est utilisé pour communiquer une liste des sujets ou des collections présents dans un répertoire. Il est tout à fait possible qu'un répertoire OAI n'ait pas d'ensemble (set), et en conséquence la réponse ne contiendra pas d'information sur des ensembles. Exemple:

<http://www.infomotions.com/alex/oai/?verb=ListSets> [<http://www.infomotions.com/alex/oai/?verb=ListSets>]

4. ListIdentifiers - On présume que dans un répertoire chaque item est associé à une sorte de clé unique, un identifiant. Ce verbe demande au répertoire la liste des identifiants. Cette liste pouvant être assez longue, et le contenu d'un répertoire pouvant changer avec le temps, on peut associer à cette commande un certain nombre de qualificatifs optionnels, dont un "jeton de reprise" (resumption token), des dates, ou des spécifications d'ensembles. En bref, cette commande demande au répertoire : "qu'avez-vous en magasin?" Exemple:

<http://www.infomotions.com/alex/oai/?verb=ListIdentifiers>
[<http://www.infomotions.com/alex/oai/?verb=ListIdentifiers>]

5. GetRecord - Ce verbe nous permet de récupérer de l'information à propos d'un enregistrement de métadonnées spécifique. Il nécessite deux qualificatifs: 1) le nom d'un identifiant, et 2) le nom du format de métadonnées dans lequel on veut que les données soient codées. Le résultat consistera en un item du répertoire OAI. Exemple:

http://www.infomotions.com/alex/oai/?verb=GetRecord&metadataPrefix=oai_dc&identifiant=twain-new-36
[http://www.infomotions.com/alex/oai/?verb=GetRecord&metadataPrefix=oai_dc&identifiant=twain-new-36]

6. ListRecords - Cette commande est une version généralisée de GetRecord. Elle permet au fournisseur de service de récupérer des données d'un répertoire OAI sans connaître d'identificateurs spécifiques. Cette commande permet de récupérer le contenu d'un répertoire en masse. Elle admet aussi des qualificatifs permettant de délimiter des ensembles de spécifications. Ce verbe doit avoir au moins un qualificatif, une spécification de métadonnée. Exemple:

http://www.infomotions.com/alex/oai/?verb=ListRecords&metadataPrefix=oai_dc
[http://www.infomotions.com/alex/oai/?verb=ListRecords&metadataPrefix=oai_dc]

Réponses -- le flux XML

Quand il reçoit l'un des verbes décrit ci-dessus, le répertoire OAI doit répondre sous forme d'un flux XML, et dans la

mesure où la communication se passe dans le cadre du protocole HTTP, l'en-tête HTTP doit porter en type de contenu (content-type) la mention text/xml. Les codes d'erreurs éventuelles sont transmis par le status-code de HTTP.

Toutes les réponses ont un format similaire. Elles commencent par une déclaration XML. La racine du flux XML répète toujours le nom du verbe envoyé dans la requête ainsi que la liste des name-spaces et leur schéma. Derrière viennent la date et une reprise de la requête initiale.

Chaque verbe attend en réponse un certain nombre d'éléments XML différents dans la réponse. Par exemple, le verbe Identify attend les éléments: repositoryName, baseURL, protocolVersion, et adminEmail. Ci-dessous une réponse simple, mais valide, au verbe Identify:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Identify
  xmlns="http://www.openarchives.org/OAI/1.0/OAI_Identify"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_Identify
http://www.openarchives.org/OAI/1.0/OAI_Identify.xsd">

  <responseDate>2002-02-16T09:40:35-7:00</responseDate>
  <requestURL>http://www.infomotions.com/alex/oai/index.php?verb=Identify</requestURL>

  <!-- Identify-specific content -->
  <repositoryName>Alex Catalogue of Electronic Texts</repositoryName>
  <baseURL>http://www.infomotions.com/alex/</baseURL>
  <protocolVersion>1.0</protocolVersion>
  <adminEmail>eric_morgan@infomotions.com</adminEmail>
</Identify>
```

Le résultat de la requête ListMetadataFormats doit être un ensemble d'informations à propos des formats de métadonnées gérés par le répertoire. Donc, la réponse à la requête ListMetadataFormats doit comprendre un élément metadataFormat avec un certain nombre de sous-éléments: metadataPrefix, schema, metadataNamespace. Voici un exemple:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListMetadataFormats
  xmlns="http://www.openarchives.org/OAI/1.0/OAI_ListMetadataFormats"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_ListMetadataFormats
http://www.openarchives.org/OAI/1.0/OAI_ListMetadataFormats.xsd">

  <responseDate>2002-02-16T09:51:49-7:00</responseDate>
  <requestURL>http://www.infomotions.com/alex/oai/index.php?verb=ListMetadataFormats</requestURL>

  <!-- ListMetadataFormats-specific content -->
  <metadataFormat>
    <metadataPrefix>oai_dc</metadataPrefix>
    <schema>http://www.openarchives.org/OAI/dc.xsd</schema>
    <metadataNamespace>http://purl.org/dc/elements/1.1/</metadataNamespace>
  </metadataFormat>
</ListMetadataFormats>
```

Le verbe ListIdentifiers peut être illustré par les exemples les plus simples. Voici une réponse à cette commande où, en dehors des réponses standard, il y a seulement en supplément un unique élément XML, identifier:


```
<?xml version="1.0" encoding="UTF-8" ?>
<ListIdentifiers
  xmlns="http://www.openarchives.org/OAI/1.0/OAI_ListIdentifiers"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_ListIdentifiers
http://www.openarchives.org/OAI/1.0/OAI_ListIdentifiers.xsd">

  <responseDate>2002-02-16T10:03:09-7:00</responseDate>
  <requestURL>http://www.infomotions.com/alex/oai/index.php?verb=ListIdentifiers</requestURL>

  <!-- ListIdentifiers-specific content -->
  <identifier>twain-30-44</identifier>
  <identifier>twain-adventures-27</identifier>
  <identifier>twain-adventures-28</identifier>
  <identifier>twain-connecticut-31</identifier>
  <identifier>twain-extracts-32</identifier>
</ListIdentifiers>
```

Le dernier exemple illustre une réponse au verbe GetRecord. Il comporte bien plus d'informations que les exemples précédents, parce qu'il s'agit vraiment du coeur de l'affaire. Les éléments XML incluent l'élément record (la notice) et ses sous-éléments tels que spécifié par le format des métadonnées:

```
<?xml version="1.0" encoding="UTF-8" ?>
<GetRecord
  xmlns="http://www.openarchives.org/OAI/1.0/OAI_GetRecord"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_GetRecord
http://www.openarchives.org/OAI/1.0/OAI_GetRecord.xsd">

  <responseDate>2002-02-16T10:09:35-7:00</responseDate>
  <requestURL>http://www.infomotions.com/alex/oai/index.php?verb=GetRecord&metadataPrefix=oai_dc&iden

  <!-- GetRecord-specific content -->
  <record>

    <header>
      <identifier>twain-tom-40</identifier>
      <datestamp>1999</datestamp>
    </header>

    <metadata>

      <!-- Dublin Core metadata -->
      <dc xmlns="http://purl.org/dc/elements/1.1/"
        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
        xsi:schemaLocation="http://purl.org/dc/elements/1.1/
http://www.openarchives.org/OAI/dc.xsd">

        <creator>Twain, Mark</creator>
        <title>Tom Sawyer, Detective</title>
        <date>1903</date>
        <identifier>http://www.infomotions.com/etexts/literature/american/1900-/twain-tom-40.txt</id
        <rights>Ce document est dans le domaine public.</rights>
        <language>en-US</language>
        <type>text</type>
        <format>text/plain</format>
        <relation>http://www.infomotions.com/alex/</relation>
        <relation>http://www.infomotions.com/alex/cgi-bin/concordance.pl?cmd=selectConcordance&bookc
        <relation>http://www.infomotions.com/alex/cgi-bin/configure-ebook.pl?handle=twain-tom-40</re
        <relation>http://www.infomotions.com/alex/cgi-bin/pdf.pl?handle=twain-tom-40</relation>
        <contributor>Morgan, Eric Lease</contributor>
        <contributor>Infomotions, Inc.</contributor>

      </dc>
```

```
</metadata>
</record>
</GetRecord>
```

Un exemple

En un après-midi j'ai créé le début d'une application de fourniture de contenu en PHP. Le code source de cette application est disponible à l'adresse <http://www.infomotions.com/alex/oai/alex-oai-1.0.tar.gz>. Ci-dessous vous trouverez un bout du code implémentant le verbe ListIdentifiers. Le fichier ListIdentifiers.php interroge la base de données (MySQL) sous-jacente au système pour obtenir une liste de clés et de données telles que définies par le protocole:

```
<?php

# Début de la réponse
echo '<ListIdentifiers'
  xmlns="http://www.openarchives.org/OAI/1.0/OAI_ListIdentifiers"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/1.0/OAI_ListIdentifiers
    http://www.openarchives.org/OAI/1.0/OAI_ListIdentifiers.xsd">';
echo '<responseDate>'. RESPONSEDATE . '</responseDate>';
echo '<requestURL>' . REQUESTURL . '</requestURL>';

# créé une requête SQL et l'exécute
$sql = "SELECT filename
  FROM titles
  WHERE filename like 'twain%'
  ORDER BY filename";
$rows = mysql_db_query (DATABASE, $sql);
checkResults();

# fait une boucle pour chaque enregistrement trouvé
while ($r = mysql_fetch_array($rows)) {

  # l'affiche
  echo '<identifier>' . $r["filename"] . '</identifier>';

}

# fin de la réponse
echo '</ListIdentifiers>';

?>
```

Conclusion

Cet article a attiré l'attention sur les objectifs du protocole Open Archives Initiative (OAI), en donnant quelques exemples. Cette introduction peut vous permettre de lire les spécifications et devenir un fournisseur de contenus. Il est plus complexe d'être un fournisseur de service. Si Google peut fournir d'excellents mécanismes de recherches pour Internet dans son ensemble, des services implémentant OAI peuvent fournir des moyens plus spécialisés pour explorer le "web caché".

Part III. Annexes

Table of Contents

A. Quelques lectures 70

 XML en général 70

 Cascading Style Sheets - CSS 70

 XSLT 70

 DocBook 70

 XHTML 71

 RDF 71

 EAD 71

 TEI 71

 OAI-PMH 71

Appendix A. Quelques lectures

Voici une courte liste de livres et de sites web qui peuvent être utilisés pour approfondir votre connaissance d'XML.

XML en général

1. XML in a Nutshell par Elliotte Rusty Harold - une vaste présentation de XML.
2. XML for the World Wide Web par Elizabeth Castro - Une introduction pas-à-pas à de nombreux aspects d'XML. Très visuel.
3. XML From the Inside Out [<http://www.xml.com/>] - un bon site plein d'articles sur XML
4. XML Tutorial [<http://www.w3schools.com/xml/>] - Testez vos connaissances d'XML ici.
5. DTD Tutorial [<http://www.w3schools.com/dtd/>] - Sur ce site, apprenez-en plus sur les DTD.
6. Extensible Markup Language (XML) [<http://www.w3.org/XML/>] - La page d'accueil officielle sur XML.
7. STG XML Validation Form [<http://www.stg.brown.edu/service/xmlvalid/>] - Utilisez cette adresse pour voir si votre XML est correct.

Cascading Style Sheets - CSS

1. Cascading Style Sheets, designing for the Web by Hakon Wium Lie - Un des ouvrages les plus complets sur XML.
2. Cascading Style Sheets [<http://www.w3.org/Style/CSS/>] - Page d'accueil officielle de CSS.
3. W3C CSS Validation Service [<http://jigsaw.w3.org/css-validator/>] - Vérifiez vos fichiers CSS ici.
4. CSS Tutorial [<http://www.w3schools.com/css/>] - Testez votre connaissance de CSS avec ce didacticiel.

XSLT

1. XSLT Programmer's Reference par Michael Kay - La référence la plus complète sur XSLT
2. Extensible Stylesheet Language (XSL) [<http://www.w3.org/Style/XSL/>] - La page d'accueil officielle de XSLT.
3. XSL Tutorial [<http://www.w3schools.com/xsl/>] - Testez ici vos connaissances sur XSLT

DocBook

1. DocDocBook, the definitive guide par Norman Walsh - Un peu daté, mais reste l'ouvrage imprimé de référence sur DocBook.
2. DocBook Open Repository [<http://docbook.sourceforge.net/>] - Le meilleur endroit du web pour trouver des

choses sur DocBook.

XHTML

1. Special Edition Using HTML and XHTML par Molly E. Holzschlag - Couvre l'ensemble de XHTML.
2. HyperText Markup Language (HTML) Home Page [<http://www.w3.org/MarkUp/>] - Page d'accueil officielle de HTML
3. Markup Validation Service [<http://validator.w3.org/>] - Vérifiez ici la validité de votre code HTML

RDF

1. Resource Description Framework (RDF) [<http://www.w3.org/RDF/>] - Page d'accueil officielle pour RDF.
2. RDF Validation Service [<http://www.w3.org/RDF/Validator/>] - Validez ici vos fichiers RDF.
3. Dublin Core Metadata Initiative [<http://www.dublincore.org/>] - Site officiel du Dublin Core.
4. Semantic Web [<http://www.w3.org/2000/01/sw/>] - Décrit les objectifs du web sémantique.

EAD

1. Encoded Archival Description (EAD) [<http://www.loc.gov/ead/>] - Page d'accueil officielle pour EAD.
2. EAD Cookbook [<http://jefferson.village.virginia.edu/ead/cookbookhelp.html>] - Un magnifique manuel pour tout ce qui concerne EAD.

TEI

1. Site web TEI [<http://www.tei-c.org/>] - Site officiel de TEI.
2. TEI Lite [<http://www.tei-c.org/Lite/>] - Une description complète de TEI Lite.
3. Feuilles de style TEI [<http://www.tei-c.org/Stylesheets/>] - Une courte liste de feuilles de styles CSS et XSLT pour TEI

OAI-PMH

1. Open Archives Initiative [<http://www.openarchives.org/>] - Site officiel OAI-PMH.