PHP-XML

Table des matières

php-xml en quelques questions :

Introduction

Pourquoi s'intéresser à XML et PHP

L'XML (eXtensible Markup Language) s'impose de plus en plus comme la solution pragmatique à de nombreux problèmes nécessitant des échanges de données structurées. C'est un outil d'intégration d'applications puissant. A partir de ce langage, il est en effet possible de créer des pages dynamiques, d'agréger le contenu de sources hétérogènes et de faire de la diffusion multi-terminaux. Son atout majeur : il sépare la source d'information de sa mise en forme.

De ce fait, dans les applications actuelles, XML peut résolument prétendre à un rôle de format pivot permettant d'alléger les développements et de renforcer leur pérennité ainsi que leur modularité.

Les points forts de XML:

- XML est aisément lisible et ne nécessite théoriquement pas de connaissances particulières pour être compris;
- XML est auto-descriptif et extensible;
- XML est basé sur une structure arborescente permettant de modéliser la majorité des problèmes informatiques;
- XML se veut " universel " et est portable;
- XML est standardisé;
- XML peut être distribué par n'importe quel protocole à même de transporter du texte (comme HTTP);
- XML est facile à intégrer ; il est utilisable par toute application pourvue d'un parseur XML (c'est-à-dire un logiciel permettant d'analyser un code XML).

De son côté, le langage de script PHP est un outil de développement spécifiquement conçu pour le Web. Il est mis en oeuvre sur un serveur web et offre la possibilité de développer des sites web dynamiques.

PHP est un produit Open Source, ce qui signifie qu'il peut être utilisé, modifié et distribué librement. PHP est également multi-plateforme, portable, et aisément programmable. Toutes ces caractéristiques en font un outil fort apprécié des développeurs Web.

Depuis la version 4, PHP intègre un certain nombre de fonctionnalités (allant croissant) orientées XML. Les programmeurs PHP peuvent donc à présent directement exploiter dans leurs scripts le langage XML et ses outils associés. Il est ainsi possible, à partir d'une application PHP, de créer des documents XML, de les modifier, de les analyser, de les valider ou de les transformer en fichier HTML, PDF ...

Intégration PHP/XML

L'intégration de la technologie XML (et outils associés) dans le langage PHP augmente de façon significative les capacités de ce dernier à dialoguer avec des systèmes hétérogènes.

Exemples d'utilisation du couple PHP/XML

- Mise en place et exploitation de "Web Services"
- Production, diffusion et récupération de données selon un format d'échange standardisé
- Modularisation et configuration des scripts sur base du format XML
- Production et exploitation de flux RSS
- Mise en forme d'information dans de multiples formats (XML, HTML, WML, PDF, CSV, SVG, etc.)

Le langage XML

XML est un métalangage. C'est un langage de balisage permettant de définir des langages spécialisés, extensibles, qui seront utilisés pour décrire des structures de données propres à un domaine applicatif.

L'objectif du langage XML est de faciliter le traitement automatisé de documents et de données. L'idée est de structurer les informations de telle manière qu'elles puissent être à la fois lues par des personnes et traitées par des applications qui exploiteront de manière automatisée les informations en question.

Les blocs d'instructions

```
1  - <?xml version="1.0" encoding="iso-8859-1"?>
2  -
3  - <bibliographie>
4  -
5  - <livre>
6  - <titre>Clandestin</titre>
7  - <auteur>James Ellroy</auteur>
8  - <editeur>Rivages</editeur>
9  - <lieu>Paris</lieu>
10 - <date>1990</date>
11 - <pages>445</pages>
12 - <description>
13 - Il n'y a pas que les enqu&ecirc;tes de police ... </description>
14 - </livre>
15 -
16 - <livre>
17 - <titre>Le Dahlia noir</titre>
18 - <auteur>James Ellroy</auteur>
19 - <editeur>Rivages</editeur>
20 - <lieu>Paris</lieu>
21 - <date>1990</date>
22 - <pages>472</pages>
23 - <description>
24 - Le 15 janvier 1947, dans un terrain vague ... </description>
25 - </livre>
26 -
27 - </bibliographie>
```

La structure de base des formats XML consiste toujours en une description par balises.

Principe de fonctionnement

Avec l'XML, on peut créer librement toutes les balises souhaitées. Pour tirer profit de cette structure, chaque application devra donner une sémantique à ces balises, c'est à dire associer un comportement à chaque type d'élément.

L'exemple précédent propose une bibliographie codée en XML. L'élément livre (dont le nom est choisi par le concepteur du document) contient différents descripteurs bibliographiques (titre, auteur, date ...). Une application pourra présenter cette bibliographie sous forme d'une liste ordonnée par nom d'auteur, alors qu'une autre application la fera apparaître sous forme d'une liste de liens renvoyant à des fiches descriptives détaillées.

Complément d'information

Une analyse plus poussée de la norme XML dépasse le cadre de ce cours. Pour plus de détails vous pouvez consulter la documentation oficielle :
- http://www.w3.org/XML/

ou les ressources en ligne aux adresses suivantes:
- http://www.w3schools.com/
- http://xmlfr.org/
- http://francexml.free.fr/
- http://www.xml.org/
- http://www.xml.com/

La manipulation du format XML

Le langage XML, pris isolément, n'est qu'un format de structuration de données. Un fichier XML ne fait que stocker, en suivant certaines règles, des informations sur disque. Ce sur quoi se fonde l'intelligence d'une application orientée XML ce sont les opérations qui vont être effectuées sur les documents : recherche, mise à jour, transformation. Pour cela, le repérage, le décodage et la récupération des informations encapsulées dans le format XML vont nécessiter le recours à des outils d'analyse adaptés.

Dans ce cadre, la manipulation d'un document XML peut se faire de différentes façons et à travers différents outils. La transformation d'XML vers le format HTML, relève habituellement d'un langage spécifique appelé XSLT. L'analyse et la mise à jour de documents XML peuvent, elles, être directement prises en charge par le langage de programmation (ici PHP) via les interfaces SAX (Simple API for XML) et DOM (Document Object Model).

- Le langage XSLT (Extensible Stylesheet Language Transformation) XSLT est un sous-ensemble du langage XSL qui permet d'effectuer des traitements et des transformations sur les données XML. Le document produit en résultat peut l'être dans différents langages (HTML, XHTML, XML, WML, texte). A partir de PHP, il est possible de lancer, de guider et de paramétrer des transformations utilisant le langage XSLT.
- L'api SAX (Simple API for XML) SAX est une interface de programmation permettant la manipulation de données au format XML. Elle est basée sur une approche événementielle, ce qui veut dire que l'analyse XML est régie par l'apparition ou non de certains événements. PHP implémente l'interface SAX.
- L'api DOM (Document Object Model) DOM est une interface spécifiée par le W3C permettant de créer, élaborer, modifier et parcourir une structure de document XML. Elle est basée sur une représentation hiérarchique des éléments XML. PHP implémente l'interface DOM.

Dans la suite de ce cours, nous verrons, à travers différentes applications concrètes, comment mettre en oeuvre ces différents outils via PHP.

XSLT

Le XSL (Extensible Style Language) est le langage utilisé pour définir les feuilles de style qui seront associées aux documents XML. Il a deux composantes : le langage de transformation des données XSLT (il permet de manipuler les données et de les réorganiser) et le langage de formatage des données XSL/FO (il permet de définir des marges, des polices, des positionnements ...). Nous ne nous intéresserons ici qu'au langage XSLT, le deuxième dépassant largement le cadre de ce cours.

Le langage XSLT définit la syntaxe et la sémantique d'une feuille de styles utilisée pour transformer un document XML en un autre document. Cette transformation peut produire un document très différent de celui d'origine (document XML, document HTML ou XHTML, document texte, document PDF, etc.).

Principe de fonctionnement

Une feuille de styles XSLT est composée d'une suite de règles appelées "template rules" (ou règles de gabarit en français).

Le processeur XSLT (composant logiciel chargé de la transformation) crée une structure logique arborescente (on parle d'arbre source) à partir du document XML. Il lui applique ensuite des transformations selon les template rules contenues dans la feuille XSLT pour produire un arbre résultat représentant, par exemple, la structure d'un document HTML.

Chaque "template rule" définit des traitements à effectuer sur un élément (noeud ou feuille) de l'arbre source. On appelle "patterns" (en français motifs, parfois "éléments cibles") les éléments de l'arbre source.

L'arbre source peut être entièrement remodelé et filtré, si bien que l'arbre résultat peut être radicalement différent de l'arbre source.

Feuille de styles XSLT

```
1  - <?xml version="1.0" encoding="ISO-8859-1"?>
2  - <xsl:stylesheet version="1.0"
3  - xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  -
5  - <xsl:template match="/">
6  - <html>
7  - <head>
8  - <title>Bibliographie</title>
9  - </head>
10 - <body bgcolor="#FFFFFF">
11 - <ul>
12 - <xsl:apply-templates/>
13 - </ul>
14 - </body>
15 - </html>
16 - </xsl:template >
17 -
18 - <xsl:template match="introduction" >
19 - <h1>
20 - Introduction
21 - </h1>
22 - <blockquote>
23 - <xsl:value-of select="."/>
24 - </blockquote>
25 - </xsl:template >
26 -
27 - <xsl:template match="livre" >
28 - <li>
29 - <xsl:value-of select="titre"/> - <xsl:value-of select="auteur"/>
```

```
30 - </li>
31 - </xsl:template >
32 -
33 - </xsl:stylesheet>
```

Ci-dessus un exemple d'une feuille de styles XSLT transformant un document XML en un document HTML. On remarque facilement les deux 'template rules' qui sont délimitées par les codes '<xsl:template>' et '</xsl:template>'. Elles contiennent chacune des instructions de transformation à apporter aux éléments XML qui sont repérés à l'aide des attributs 'match'.

Principales instructions XSLT

`<xsl:stylesheet>`
Déclaration de la feuille de style. Tout ce qui se situe à l'intérieur de cette balise relève le la logique de la feuille de style.

`<xsl:output method='html'/>`
L'élément <xsl:output> indique le format de sortie du document résultant. Il est à placer immédiatement après l'élément <xsl:stylesheet>

`<xsl:template match="/">`
L'élément <xsl:template> permet de définir une règle de modèle. Ici, par l'intermédiaire de l'attribut 'match' c'est à la racine ('/') du document que l'on applique des règles de style.

`<xsl:apply-templates/>`
L'élément <apply-templates> permet d'appliquer les modèles (templates) d'une feuille de style sur les fils et les noeuds de type texte du noeud courant.

`<xsl:call-template name="...">`
L'élément <xsl:call-template> permet d'appeler un modèle (template) directement par son nom.

`<xsl:value-of select="..."/>`
L'élément <xsl:value-of> permet d'extraire la valeur d'un noeud sélectionné via l'attribut 'select'.

Résultat de la transformation

```
1  - <html>
2  - <head>
3  - <title>Bibliographie</title>
4  - </head>
5  - <body bgcolor="#FFFFFF">
6  - <h1>
7  - Introduction
8  - </h1>
9  - <blockquote>
10 - Cette bibliographie rassemble les lvres de ...
11 - </blockquote>
12 - <ul>
13 - <li>
14 - La Bête hors des âges - Henri Vernes
15 - </li>
16 - <li>
17 - Barberousse - Michel Tournier
18 - </li>
19 - <li>
20 - Brown's requiem - James Ellroy
21 - </li>
22 - </ul>
23 - </body>
24 - </html>
```

On peut constater ci-dessus que la liste à puces HTML a bien été constituée à partir de l'application de la template rule "livre" précédemment définie dans la feuille XSLT.

XPATH

XPath est un langage d'expressions qui permet de sélectionner des éléments ou des groupes d'éléments au sein d'un document XML. Il est notamment exploité par XSLT lors de la procédure de transformation d'un document XML.

Le point de départ d'Xpath est la modélisation d'un document XML sous la forme d'un arbre composé d'un ensemble de noeuds de plusieurs types :

- un noeud racine (root node), noté " / ", qui représente l'entièreté du document;
- des noeuds éléments (pour chaque élément du document);
- des noeuds attributs (pour chaque attribut au sein d'un élément);
- des noeuds textes (pour chaque données textuelles ou la valeur d'un attribut);
- des noeuds commentaires (pour chaque commentaire dans le document);
- des noeuds instructions de traitement (pour chaque instruction de traitement dans le document);
- des noeuds namespaces (pour chaque namespace utilisé dans le document).

Cette modélisation va être exploitée par XPATH afin de rendre possible la sélection de sous-ensembles de noeuds. Sans cette possibilité de " ciblage " aucun traitement conséquent de l'information XML n'est possible.

Les expressions XPATH

XPATH est formulé et utilisé sous forme d'expressions. Ces expressions, après avoir été évaluées, peuvent fournir en réponse les éléments suivants :

- un ensemble de noeuds;
- un booléen;
- un nombre;
- une chaîne de caractères.

Les expressions Xpath servent en fait à définir ce que l'on appelle couramment des "patterns", c'est-à-dire des chaînes de caractères permettant de repérer un noeud dans un document XML (on peut faire le parallèle avec la technique des expressions régulières). Voici quelques exemples de patterns.

> **Les "patterns"**
>
> `nom`
> exemple : livre
> description : sélectionne des éléments en fonction de leur nom (ici 'livre')
>
> `nom[ ]`
> exemple : livre[1]
> description : sélectionne un élément ayant un nom donné en fonction de son ordre d'apparition (ici le premier élément 'livre')
>
> `nom[end]`
> exemple : livre[end]
> description : sélectionne le dernier élément ayant un nom donné (ici le dernier élément 'livre')
>
> `|`
> exemple : livre|revue
> description : Indique une alternative (l'élément 'livre' ou l'élément 'revue')
>
> `/`

exemple : /
description : utilisé seul, ce pattern définit l'élément de plus haut niveau de l'arborescence ainsi que (de façon implicite) tous ses éléments fils.
exemple : bibliographie/livre
description : utilisé entre deux éléments, il décrit la localisation d'un élément dans son arborescence

`*`
exemple : *
description : désigne n'importe quel élément

`//`
exemple : //livre
description : désigne tous les descendants d'un noeud

`.`
exemple : .
description : désigne le noeud courant

`..`
exemple : ..
description : désigne le noeud parent

`@`
exemple : @titre
description : désigne un attribut (la notation @* désigne tous les attributs d'un élément)

`text()`
exemple : text()
description : désigne le contenu d'un élément (le texte contenu entre ses balises)

Exemples d'expressions XPATH

Si on veut sélectionner l'ensemble des noeuds <livre> qui sont enfants du noeud <bibliographie> (l'élément racine du document), on écrira :
- /biblographie/livre

Si on veut sélectionner tous les noeuds <livre> dont le lieu d'édition est Paris, on écrira :
- /biblographie/livre[lieu='Paris']

Si on veut sélectionner tous les noeuds <nom> des auteurs qui ont la nationalité belge on écrira :
- /biblographie/livre/auteur[@pays="Belgique"]/nom

Ce qui se trouve entre crochets s'appelle un prédicat ; il représente une condition qui doit être vérifiée pour sélectionner un noeud.

Les axes de recherche

Les expressions XPATH peuvent faire référence à des axes de recherche. Ces axes de recherche précisent dans quel sens il faut parcourir l'arbre afin de sélectionner des noeuds.

Les différents axes de recherche

`child::`
Clause sollicitant les enfants du noeud contexte - c'est l'axe par défaut.

`descendant::`
Clause sollicitant les descendants du noeud contexte (les enfants, les enfants des enfants, etc.).

`cdescendant-or-self::`
Clause sollicitant les descendants du noeud contexte et le noeud contexte lui-même.

`parent::`
Clause sollicitant le parent du noeud contexte.

`ancestor::`
Clause sollicitant les ancêtres du noeud contexte (le parent, le parent du parent, etc.).

`ancestor-or-self::`
Clause sollicitant les ancêtres du noeud contexte et le noeud contexte lui-même.

`following-sibling::`
Clause sollicitant les frères du noeud contexte (qui ont le même parent) et qui suivent celui-ci dans le contexte.

`preceding-sibling::`
Clause sollicitant les frères du noeud contexte (qui ont le même parent) et qui précèdent celui-ci dans le contexte.

`following::`
Clause sollicitant tous les noeuds qui suivent le noeud contexte dans l'ordre du document, à l'exclusion des descendants du noeud contexte (et des noeuds attributs et namespaces).

`preceding::`
Clause sollicitant tous les noeuds qui précèdent le noeud contexte dans l'ordre du document, à l'exclusion des ancêtres du noeud contexte (et des noeuds attributs et namespaces).

`attribute::`
Clause sollicitant tous les noeuds attributs enfants du noeud contexte.

`namespace::`
Clause sollicitant tous les noeuds namespaces enfants du noeud contexte.

`self::`
Clause sollicitant le noeud contexte lui-même.

Exemples d'axes de recherche

L'expression '/descendant::*' va sélectionner tous les éléments à partir de la racine du

document.

L'expression 'ancestor::chapitre' va sélectionner tous les ancêtres de l'élément contexte ayant pour nom chapitre.

L'expression 'child::adresse/child::rue' va sélectionner les enfants rue des enfants adresse de l'élément contexte. Elle peut être résumée par 'adresse/rue'.

L'expression 'parent::node()/child::titre' va sélectionner les enfants titre de l'élément parent de l'élément contexte. Elle peut être résumée par '../titre'.

L'expression 'child::personne[attribute::titre="M"]' va sélectionner les enfants personne (si l'attribut titre est égal à M) de l'élément contexte. Elle peut être résumée par 'personne[@titre="M"]'.

Nous verrons par la suite comment, dans une procédure PHP, il est possible de faire appel aux expressions XPATH pour exploiter les données encodées au format XML.

API SAX

SAX ("Simple API for XML" ou en français "API simple pour XML") est une interface de programmation qui permet l'analyse du format XML. Elle a été mise au point par David Megginson et est progressivement devenue un standard de fait.

La particularité de SAX, c'est qu'elle traite un document XML comme un flux. Ainsi, cette API fournit à l'application qui l'utilise les éléments constitutifs du document au fur et à mesure de la lecture. On dit qu'il s'agit d'une approche de type événementiel.

Une application qui exploite l'API SAX va, en conséquence, mettre en oeuvre des gestionnaires d'événements. A mesure que l'analyse du document XML se poursuit, des événements sont relevés et signifiés au programme qui peut ensuite les traiter. Par exemple, un événement peut être généré pour chaque nouvelle ouverture de balise rencontrée, ou pour chaque fermeture de balise.

Approche événementielle SAX

Exemple de document XML:

```
<?xml version="1.0"?>
<auteurs>
  <nom>James Ellroy</nom>
  <nom>Jack London</nom>
  <nom>Jules Vernes</nom>
</auteurs>
```

Analyse SAX :

```
start document
    |
    |-- start element 'auteurs'
    |       |
    |       |-->start element 'nom'
    |       |       |
    |       |       |-->characters 'James Ellroy'
    |       |
    |       |-->end element 'nom'
    |       |
    |       |-->start element 'nom'
    |       |       |
    |       |       |-->characters 'Jack London'
    |       |
    |       |-->end element 'nom'
    |       |
    |       |-->start element 'nom'
    |       |       |
    |       |       |-->characters 'Jules Vernes'
    |       |
    |       |-->end element 'nom'
```

```
    |
    |-- end element 'auteurs'
    |
end document
```

L'analyse SAX repose sur la prise en compte d'événements : début du document, fin du document, début d'un élément, fin d'un élément ... C'est au programmeur à tirer profit de l'apparition ou non de ces événements afin de retirer du document l'information qui l'intéresse.

De nombreux langages de programmation intègrent aujourd'hui l'interface SAX : Java, C, C++, Perl, PHP, Python ...

SAX et les outils qui lui sont associés sont souvent mis en avant pour l'efficacité et la rapidité de l'analyse XML qu'ils fournissent. On reconnaît toutefois également que cette api est plus lourde à manipuler pour le programmeur. C'est à ce stade de la réflexion qu'il faut envisager l'autre API de manipulation XML : DOM.

API DOM

DOM ("Document Object Model" ou en français "modèle objet de documents") est une spécification du W3C. Elle définit une interface de programmation qui permet à des applications informatiques d'accéder à une représentation "orientée objet" des documents XML et HTML. Cette interface est indépendante de toute plate-forme et de tout langage.

Avec DOM, l'accès aux éléments constitutifs d'un document est fortement facilité. Les programmeurs peuvent, en effet, manipuler et parcourir de façon arborescente un document.

DOM considère un document comme étant constitué de noeuds et les traite en tant qu'objets. Ceux-ci sont de différents types : le document, les éléments, les attributs, les commentaires, la déclaration, les contenus textuels. L'interface DOM, à travers la définition de propriétés et de méthodes, donne accès à tous ces éléments et à leur contenu. Il est ainsi fort aisé de modifier, supprimer, créer, réécrire des documents XML.

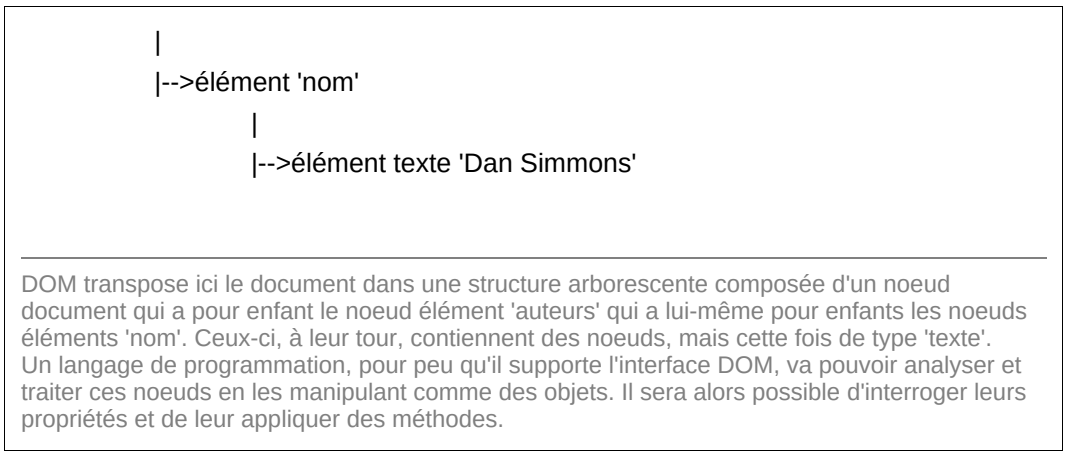
Modéle de document DOM

Exemple de document XML:

```
<?xml version="1.0"?>
<auteurs>
  <nom>James Ellroy</nom>
  <nom>Jack London</nom>
  <nom>Jules Vernes</nom>
  <nom>Umberto Eco</nom>
  <nom>Dan Simmons</nom>
</auteurs>
```

Représentation DOM :

```
document
    |
    |--élément 'auteurs'
            |
            |-->élément 'nom'
            |       |
            |       |-->élément texte 'James Ellroy'
            |
            |-->élément 'nom'
            |       |
            |       |-->élément texte 'Jack London'
            |
            |-->élément 'nom'
                    |
                    |-->élément texte 'Jules Vernes'
            |
            |-->élément 'nom'
                    |
                    |-->élément texte 'Umberto Eco'
```



DOM transpose ici le document dans une structure arborescente composée d'un noeud document qui a pour enfant le noeud élément 'auteurs' qui a lui-même pour enfants les noeuds éléments 'nom'. Ceux-ci, à leur tour, contiennent des noeuds, mais cette fois de type 'texte'. Un langage de programmation, pour peu qu'il supporte l'interface DOM, va pouvoir analyser et traiter ces noeuds en les manipulant comme des objets. Il sera alors possible d'interroger leurs propriétés et de leur appliquer des méthodes.

De nombreux langages de programmation ont implémanté l'interface DOM et s'ouvrent de cette façon au développement orienté XML : Java, C, C++, Perl, PHP, Python, JavaScript, ECMAScript. Cette implémentation se matérialise par un objet ou une librairie.

Quels outils utiliser et quand ?

Les questions s'imposent évidemment. Faut-il transformer un document à l'aide d'XSLT? Pourquoi ne pas le faire directement en DOM? SAX est-elle l'API la plus efficace pour parcourir un document XML? Quel outil utiliser pour mettre à jour un document XML? Voici quelques éléments de réponse qui vous aideront à formuler une solution adaptée à votre contexte :

XSLT

L'utilisation d'XSLT se justifie amplement lorsque :

- la transformation du document XML est conséquente
- l'objectif est la publication sous un autre format
- le besoin de programmation directe ne se fait pas sentir

SAX

L'utilisation de SAX se justifie amplement lorsque :

- le document xml à traiter est fort volumineux
- la rapidité du traitement est prioritaire
- l'objectif du parsing XML est la sélection d'informations bien ciblées
- le traitement peut être réalisé en une seule passe

DOM

L'utilisation de DOM se justifie amplement lorsque :

- des modifications doivent être effectuées dans le document XML
- la structure XML doit être exploitée finement
- l'objectif est de créer un document XML

Prérequis et installation des modules de base

Environnement Web/PHP

Afin d'exploiter le langage PHP, il faut disposer d'un compte sur un serveur web équipé du module PHP (le plus récent de préférence). Nous supposerons que c'est le cas - sinon, vous serez peut-être amené à installer cet environnement de développement (pour cela nous vous renvoyons au chapitre suivant).

La configuration que nous avons utilisée pour cet atelier est la suivante

- Plateforme Windows XP
- Serveur Apache 1.3.x
- Module PHP 4.x et 5.x

Il est bien évidemment possible de travailler de la même façon avec une configuration différente (plateforme Unix/Linux, ou serveur Web IIS par exemple). Nous ne pouvons cependant, dans le cadre de cet atelier, couvrir toutes les solutions envisageables.

Installation de PHP 4

Installation de PHP 4 sur un OS de type Windows

Première étape :

Télécharger et installer un logiciel serveur web (si ce n'est déjà fait).

Vous avez le choix :

- Apache
- Internet Information Server
- Personal Web Server
- iPlanet
- Omni HTTPd
- Webten
- Webstar
- ...

Si l'on envisage la chose sous l'angle de la portabilité et de la stabilité, l'association du logiciel serveur web 'Apache' et du module de programmation 'PHP' est certainement la solution à privilégier.

La mise en place du serveur Apache sur un système Windows est une opération relativement simple. La distribution Windows contient en effet un exécutable qui permet une installation rapide et transparente.

Deuxième étape :

Télécharger -dans un répertoire de votre choix- la version 4 de la distribution de PHP (pour Windows 95/98/NT/2000/XP) à partir de l'adresse suivante http://www.php.net/ ou depuis un des nombreux sites miroirs.

Troisième étape :

Installer sur le serveur le module PHP.

Cette opération consiste -dans un premier temps- à décompresser la distribution PHP qui vient d'être téléchargée (dans le répertoire c:\php par exemple).

Ensuite, deux fichiers (se trouvant dans le répertoire d'intsallation 'c:\php') doivent être copiés sur le sytème. Le fichier 'php4ts.dll' et 'php.ini' sont en effet indispensables au bon fonctionnement de PHP; il faut donc s'assurer qu'ils soient accessibles sur le système.

- Le fichier 'php4ts.dll' doit être placé dans le répertoire 'system': - c:\windows\system pour Windows 9x/Me - c:\winnt\system32 pour Windows NT/2000 - c:\windows\system32 pour Windows XP
- Le fichier 'php.ini-dist' doit être copié, renommé en 'php.ini' et placé dans le répertoire 'windows': - c:\windows pour Windows 9x/Me/XP - c:\winnt ou c:\winnt40 pour Windows NT/2000

Quatrième étape :

Effectuer les modifications indispensables dans le fichier de configuration de PHP (php.ini).

Dans la plupart des cas, cette opération consiste à éditer le fichier php.ini afin de préciser le chemin d'accès du répertoire où sont installées les extensions de php.

Configuration du fichier 'php.ini'

Dans la rubrique 'Paths and Directories' il faut préciser pour la propriété 'extension_dir' le chemin d'accès du répertoire qui contient les fichiers d'extensions.

```
1  - extension_dir = c:\php\extensions\
```

Il est évidemment nécessaire d'adapter le chemin d'accès en fonction du répertoire d'installation de la distribution PHP.

Cinquième étape :

Configurer le serveur web de façon à ce que le module php soit pris en compte lors des requêtes http.

En ce qui concerne le serveur Web Apache, dans la plupart des cas, il suffit d'éditer le fichier 'httpd.conf' (qui se trouve dans le répertoire 'conf' de l'installation d'Apache) et d'y ajouter les lignes de configuration tel que présenté dans l'exemple ci-dessous.

Configuration du fichier 'httpd.conf'

Les rubriques 'LoadModule', 'AddModule' et 'AddType' doivent être respectivement complétées des lignes suivantes:

```
1  - LoadModule php4_module c:/php/sapi/php4apache.dll
2  - AddModule mod_php4.c
3  - AddType application/x-httpd-php .php .phtml
```

La première ligne de configuration indique à Apache où se trouve le module PHP à charger en mémoire lors du démarrage du serveur. La deuxième ligne ne doit être insérée que si la directive 'ClearModuleList' est définie dans le fichier de configuration (au niveau de la rubrique 'AddModule' - ce qui n'est par exemple pas la cas pour Apache 2). La troisième ligne définit l'extension des fichiers PHP.

Documentation technique:

Les guides d'installation correspondant à chaque plate-forme et serveur web sont disponibles sur le site web de PHP à l'adresse suivante :
http://www.php.net/manual/installation.php

Installation de PHP 5

Afin d'exploiter le langage PHP, il faut disposer d'un compte sur un serveur web équipé du module PHP. Si ce n'est pas le cas, vous serez peut-être amené à installer cet environnement de développement. Pour cela vous devrez installer et configurer un serveur web ainsi que le module PHP. A ce niveau les options et choix possibles étant tellement nombreux, nous ne pouvons ici que formuler les grandes lignes de la procédure (pour les détails techniques et les dernières mises à jour, nous vous invitons à consulter l'information diffusée sur le site de PHP: http://www.php.net/manual/installation.php.

Installation de PHP sur un OS de type Windows

Première étape :

Télécharger et installer un logiciel serveur web (si ce n'est déjà fait).

Vous avez le choix :

- Apache
- Internet Information Server
- Personal Web Server
- iPlanet
- Omni HTTPd
- Webten
- Webstar
- ...

Si l'on envisage la chose sous l'angle de la portabilité et de la stabilité, l'association du logiciel serveur web 'Apache' et du module de programmation 'PHP' est certainement la solution à privilégier.

La mise en place du serveur Apache sur un système Windows est une opération relativement simple. La distribution Windows contient en effet un exécutable qui permet une installation rapide et transparente.

Deuxième étape :

Télécharger -dans un répertoire de votre choix- la dernière version de la distribution de PHP (pour Windows 95/98/NT/2000/XP) à partir de l'adresse suivante http://www.php.net/ ou depuis un des nombreux sites miroirs.

Troisième étape :

Installer sur le serveur le module PHP.

Cette opération consiste -dans un premier temps- à décompresser la distribution PHP qui vient d'être téléchargée (dans le répertoire c:\php par exemple).

Ensuite, deux fichiers (se trouvant dans le répertoire d'intsallation 'c:\php') doivent être copiés sur le sytème. Le fichier 'php5ts.dll' et 'php.ini' sont en effet indispensables au bon fonctionnement de PHP; il faut donc s'assurer qu'ils soient accessibles sur le système.

- Le fichier 'php5ts.dll' doit être placé dans le répertoire 'system': - c:\windows\system pour Windows 9x/Me

- c:\winnt\system32 pour Windows NT/2000 - c:\windows\system32 pour Windows XP
- Le fichier 'php.ini-dist' doit être copié, renommé en 'php.ini' et placé dans le répertoire 'windows': - c:\windows pour Windows 9x/Me/XP - c:\winnt ou c:\winnt40 pour Windows NT/2000

Quatrième étape :

Effectuer les modifications indispensables dans le fichier de configuration de PHP (php.ini).

Dans la plupart des cas, cette opération consiste à éditer le fichier php.ini afin de préciser le chemin d'accès du répertoire où sont installées les extensions de php.

Configuration du fichier 'php.ini'

Dans la rubrique 'Paths and Directories' il faut préciser pour la propriété 'extension_dir' le chemin d'accès du répertoire qui contient les fichiers d'extensions.

```
1  - extension_dir = c:\php\ext\
```

Il est évidemment nécessaire d'adapter le chemin d'accès en fonction du répertoire d'installation de la distribution PHP.

Cinquième étape :

Configurer le serveur web de façon à ce que le module php soit pris en compte lors des requêtes http.

En ce qui concerne le serveur Web Apache, dans la plupart des cas, il suffit d'éditer le fichier 'httpd.conf' (qui se trouve dans le répertoire 'conf' de l'installation d'Apache) et d'y ajouter les lignes de configuration tel que présenté dans l'exemple ci-dessous.

Configuration du fichier 'httpd.conf'

Les rubriques 'LoadModule', 'AddModule' et 'AddType' doivent être respectivement complétées des lignes suivantes:

```
1  - LoadModule php5_module c:/php/php5apache.dll
2  - AddModule mod_php5.c
3  - AddType application/x-httpd-php .php .phtml
```

La première ligne de configuration indique à Apache où se trouve le module PHP à charger en mémoire lors du démarrage du serveur. La deuxième ligne ne doit être insérée que si la directive 'ClearModuleList' est définie dans le fichier de configuration (au niveau de la rubrique 'AddModule' - ce qui n'est par exemple pas la cas pour Apache 2). La troisième ligne définit l'extension des fichiers PHP.

Sixième étape :

Vérifier si l'opération s'est bien déroulée. Pour cela vous pouvez tester le script suivant (fichier " installation_test.php ") :

Fichier " installation_test.php "

```
1  - <?php
2  - phpinfo()
3  - ?>
```

Cette fonction affiche les informations sur la configuration courante du module PHP. Pour exécuter le script présenté ci-dessus, il suffit de l'appeler via un navigateur Web en précisant dans l'url l'adresse du serveur Web sur lequel est stocké le script (ex: http://localhost/ ou http://127.0.0.1/) ainsi que le chemin d'accès au script (ex: http://localhost/mes-scripts/installation_test.php ou http://127.0.0.1/test/installation_test.php).

L'utilisation de la fonction phpinfo() a pour résultat l'affichage d'informations concernant la configuration courante de PHP: options de compilation, extensions, version, environnement, utilisateur... C'est un bon moyen de vérifier si le module PHP tourne correctement.

Documentation technique:

Les guides d'installation correspondant à chaque plate-forme et serveur web sont disponibles sur le site web de PHP à l'adresse suivante :
http://www.php.net/manual/installation.php

Installation des outils XML/XSLT sous PHP4

Parseur externe

Il s'agit ici d'utiliser, à partir de PHP, des ressources externes. La procédure d'installation va donc se limiter aux instructions d'installation de l'outil que l'on aura choisi d'utiliser.

Pour ce cours, nous ferons, dans un premier temps, appel au parseur XSLT (outil de transformation d'un document XML) de James Clark 'XT'. Il est diffusé gratuitement, il est souple, rapide et très fonctionnel. XT peut être utilisé en tant que Servlet Java, être intégré dans une application Java, ou encore être sollicité comme application indépendante.

En ce qui nous concerne, l'installation d'XT nécessite simplement le téléchargement d'un fichier exécutable pour Windows ('xt.exe' - http://www.jclark.com/xml/). Il suffit de placer celui-ci sur le système et de bien noter l'endroit où il est enregistré. Pour y accéder via PHP, seul le chemin d'accès est en effet nécessaire.

Une fois le principe de l'utilsation d'un programme externe assimilé, nous passerons en revue l'appel à d'autres outils de transformation XSLT, notamment 'msxsl' de Microsoft ainsi que le parseur 'saxon' de Michael Kay.

Installation des outils XML/XSLT sous PHP4

Sablotron

Sablotron est une boîte à outils XML implémentant les normes XSLT 1.0, DOM Level2 et XPath. C'est un logiciel libre développé et distribué par "Ginger Alliance" (http://www.gingerall.com/). La version actuelle est disponible pour différentes plate-formes: Linux, Windows NT/2000/XP, Solaris, HP-UX, Irix, FreeBSD, OpenBSD, OpenServer, Open Unix, MacOS X.

Au niveau de la plate-forme PHP 4, Sablotron a longtemps constitué l'implémentation officielle d'un parseur XSLT sous forme d'extension.

Installation sous Windows

Pour pouvoir exploiter Sablotron sous PHP 4.x vous devez vous assurer que les fichiers suivants sont présents sur votre système:

- expat.dll
- sablot.dll
- iconv.dll
- php_xslt.dll

La procédure d'installation est relativement simple. Voici la marche à suivre.

- Les fichiers 'expat.dll', 'sablot.dll' et 'iconv.dll' se trouvent dans le sous-répertoire 'dlls' du répertoire d'installation de php. Vous devez les copier dans votre répertoire 'system' : - c:\windows\system pour Windows 9x/Me - c:\winnt\system32 pour Windows NT/2000 - c:\windows\system32 pour Windows XP
- Le fichier 'php_xslt.dll' se trouve dans le sous-répertoire 'extensions' du répertoire d'installation de php (c'est son emplacement par défaut). Vérifiez qu'il s'y trouve bien.

Il faut ensuite effectuer une modification indispensable dans le fichier de configuration de PHP (php.ini). Cette opération consiste à éditer le fichier php.ini afin d'activer l'extension php_xslt.dll

Configuration du fichier 'php.ini'

Dans la rubrique 'Dynamic Extensions' il faut supprimer le signe ';' qui se trouve au début de la ligne ';extension=php_xslt.dll' (ce signe met en effet cette commande en commentaire).

```
1  - extension=php_xslt.dll
```

Cette instruction ne sera correctement prise en compte que si les fichiers "dll" précités se trouvent bien dans votre système.

Installation des outils XML/XSLT sous PHP4

DOM XML

L'extension DOM XML a été développée -à l'époque de PHP 4- afin de mettre en oeuvre l'api DOM telle que définie par le standard du W3C. Elle permet de créer et d'accéder des documents XML selon les recommandations DOM. Combinée aux librairies libxslt et libexslt, elle permet également de transformer un document XML à travers des procédures XSLT. Dans la version 5 de PHP, elle a été remplacée par l'extension DOM.

Installation sous Windows

Pour pouvoir exploiter les outils DOM XML sous PHP 4 vous devez vous assurer que les fichiers suivants sont présents sur votre système :

- libxml2.dll
- libexslt.dll
- libxslt.dll

Il est possible de se procurer ces librairies à l'adresse suivante : 'http://www.zlatkovic.com/projects/libxml/binaries.html'. Après téléchargement, les trois fichiers mentionnés doivent être copiés dans votre répertoire 'system' :
- c:\windows\system pour Windows 9x/Me
- c:\winnt\system32 pour Windows NT/2000
- c:\windows\system32 pour Windows XP

Il faut ensuite effectuer une modification indispensable dans le fichier de configuration de PHP (php.ini). Cette opération consiste à éditer le fichier php.ini afin d'activer l'extension php_domxml.dll

Configuration du fichier 'php.ini'

Dans la rubrique 'Dynamic Extensions' il faut supprimer le signe ';' qui se trouve au début de la ligne ';extension=php_domxml.dll' (ce signe met en effet cette commande en commentaire).

```
1  - extension=php_domxml.dll
```

Cette instruction ne sera correctement prise en compte que si les fichiers "dll" précités se trouvent bien dans votre système.

Installation des outils XML/XSLT sous PHP5

Parseur externe

Comme il s'agit ici d'utiliser, à partir de PHP, des ressources externes, la procédure d'installation va se limiter aux instructions d'installation de l'outil que l'on aura choisi d'exploiter. Il n'y a donc pas de différence par rapport à ce qui doit être fait sous PHP4 ni aucune liaison avec la version du module PHP.

Installation des outils XML/XSLT sous PHP5

Extension XSL

L'extension XSLT (exploitée à travers le module Sablotron) présente dans PHP 4.x a été retirée de la version 5. Pour bénéficier du support XSLT avec PHP 5, il faut à présent utiliser l'extension XSL.

L'extension XSL implémente le standard XSL, et fait des transformations XSLT à l'aide de la bibliothèque libxslt (http://xmlsoft.org/XSLT/). PHP 5 inclut l'extension XSL par défaut et peut être activée très simplement -sous Windows- en enlevant le signe de commentaire ';' devant la directive 'extension=php_xsl.dll' dans le fichier 'php.ini'. Ensuite, il est bien évidemment nécessaire de redémarrer le serveur Web.

```
extension=php_xsl.dll
```

Activation d'XSL sous Unix/Linux

L'activation de cette extension sous Unix/Linux implique la compilation préalable de libxslt et de libxml (http://xmlsoft.org/XSLT/). Une fois ces produits installés, il faut recompiler le module PHP avec l'option '--with-xsl'.

Activation d'XSL sous Unix/Linux

```
1  - # Compilation de libxml :
2  - ./configure --prefix=<DIR>
3  - make
4  - make install
```

```
1  - # Compilation de libxslt :
2  - ./configure ... --with-libxml-prefix=<DIR-libxml>
3  - make
4  - make install
```

```
1  - # Compilation de PHP :
2  - ./configure ... --with-xsl=<DIR-libxslt>
3  - make
4  - make install
```

Installation des outils XML/XSLT sous PHP5

Extension DOM

L'extension DOM XML (exploitée à travers les bibliothèques libxml et libxslt) présente dans PHP 4.x a été retirée de la version 5. Pour bénéficier du support DOM XML avec PHP 5, il faut à présent utiliser l'extension DOM.

L'extension DOM permet de manipuler des documents XML avec l'API DOM. PHP 5 inclut et active l'extension DOM par défaut.

Installation des outils XML/XSLT sous PHP5

Extension SimpleXML

L'extension SimpleXML fait partie des nouveautés offertes par PHP 5. Elle se distingue des outils envisagés jusqu'ici par sa facilité d'utilisation et permet de manipuler sans prérequis particulier des documents xml simples. L'exploitation de cette extension ne nécessite que très peu de code et de connaissances.

PHP 5 inclut et active l'extension SimpleXML par défaut.

Mise en pratique

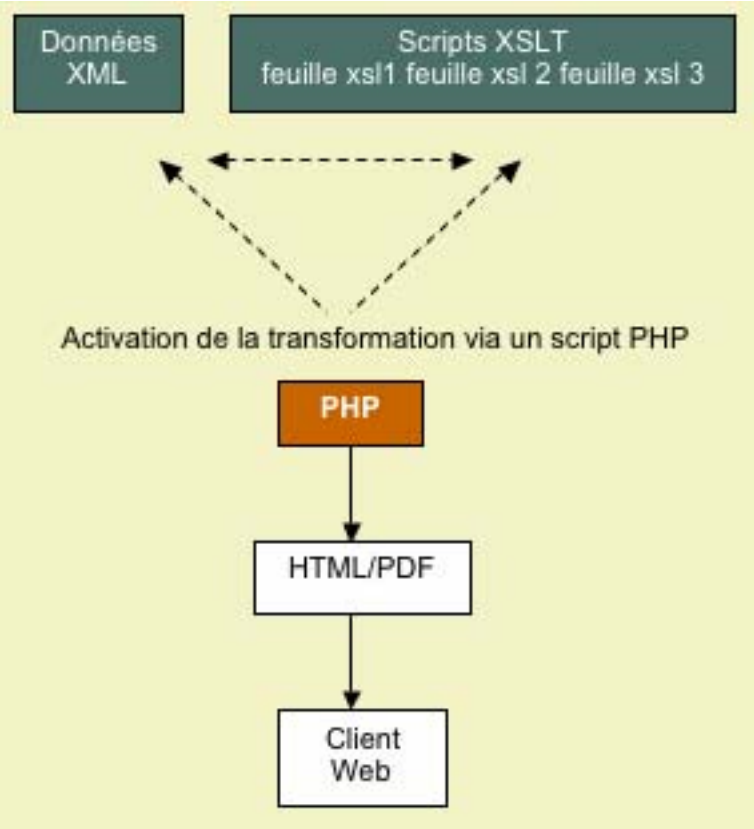
Transformation XSLT

Comment obtenir une transformation XSLT ?

Le recours à la technologie XSLT présente l'avantage d'éloigner encore un peu plus la phase de publication et d'habillage des données de la programmation proprement dite. PHP se limitant dans ce cadre à préparer l'information source et à lui appliquer des feuilles de style de façon contextuelle.

Cette répartition des tâches débouche sur un gain de temps et sur un cloisonnement qui peuvent difficilement être négligés.

PHP et XSLT



PHP va être chargé de l'orchestration des transformations XSLT.

Pour activer une transformation XSLT, c'est-à-dire l'application d'une feuille de style XSL à un document XML, il faut soit faire appel à un programme externe, soit initialiser et exploiter un processeur XSLT via les extensions PHP.

Le recours à un programme externe (par exemple 'xt', 'msxsl' ou 'saxon') est certainement la méthode la plus simple mais aussi la plus lente. L'utilisation des extensions directement prises en charge par PHP suppose plus d'efforts de codage de la part du programmeur, mais les gains en termes d'efficacité et de rapidité sont appréciables. Les chapitres qui suivent illustrent ces différentes méthodes.

Transformation XSLT

Parseur externe (PHP4 / PHP5)

La portée d'un script PHP ne se limite pas obligatoirement aux outils prévus par le langage. Il y a en effet moyen de prolonger les fonctionnalités de base en utilisant des commandes d'exécution de programmes externes.

C'est ce que nous cherchons ici pour obtenir une transformation XSLT. Nous nous en remettons, dans le cadre de cet atelier, au parseur 'xt' de James Clark. Pour l'activer, la procédure est fort simple puisqu'il suffit de lancer son exécution à patir d'un script. L'instruction d'exécution que nous utilisons est 'passthru'. Celle-ci exécute un programme externe et affiche le résultat.

Syntaxe : `passthru(command)`

Transformation de base

La transformation la plus simple consiste à appliquer une feuille de style XSL à un fichier XML. En d'autres mots, il s'agit d'utiliser le programme 'xt' en lui passant comme paramètres le chemin d'accès d'un fichier XML ainsi que celui d'un fichier XSL et d'en récupérer le résultat.

Transformation à partir du parseur 'xt'

```
1  - <?php
2  - passthru("xt.exe test.xml style1.xsl");
3  - ?>
```

Le script ci-dessus exécute le programme xt. Celui-ci associe un fichier xml à une feuille de style xsl. On suppose dans cet exemple que l'exécutable 'xt.exe', le fichier XML et le fichier XSL se trouvent dans le même répertoire que le script PHP.
Le résultat de la transformation est automatiquement redirigé vers la sortie standard.

Transformation à partir du parseur 'msxsl'

```
1  - <?php
2  - passthru("c:\msxml4\msxsl.exe test.xml style1.xsl");
3  - ?>
```

La procédure d'appel du programme externe est identique à la précédente. On suppose dans ce script que l'exécutable 'msxsl.exe' se trouve dans le répertoire 'C:\msxml4'.
Pour pouvoir utiliser ce parseur de cette façon, il faut au préalable avoir installé le module XML de Micorosft (MSXML) ainsi que l'utilitaire 'msxsl.exe'.
Plus d'information sur MSXML: http://msdn.microsoft.com/xml/

Transformation à partir du parseur 'saxon'

```
1  - <?php
2  - passthru("java -jar c:\\saxon8\\saxon8.jar test.xml style1.xsl");
3  - ?>
```

La procédure d'appel est cette foi légèrement différente puisqu'il s'agit d'appeler un programme Java. On invoque donc le moteur 'java' en lui précisant -grâce à l'option jar- où trouver les classes principales du parseur 'saxon'.
Pour pouvoir utiliser ce parseur, il faut disposer sur son système d'un environnement d'exécution Java (J2SE).
Plus d'information sur Saxon : http://www.saxonica.com/

Transformation avec passage de paramètres

Dans ce cas, l'association de la feuille de style au document XML et l'appel au parseur s'accompagnent d'une transmission de paramètres. Ceux-ci fournissent à la feuille de style des informations supplémentaires qui vont orienter la transformation du document XML. Les paramètres sont transmis sous forme de couple "nom_du_paramètre=valeur_du_paramètre".

Transformation avec passage de paramètres

```
1  - <?php
2  - # fichiers de travail
3  - $xmlfile = "../xml/biblio.xml";
4  - $xslfile = "../xsl/biblio-sablot.xsl";
5  -
6  - # commande externe
7  - passthru("xt.exe $xmlfile $xslfile titre=Biblio");
8  - ?>
```

Le script ci-dessus exécute le programme xt en lui passant le nom du fichier XML à traiter, le nom du fichier XSL à utiliser et un paramètre intitulé 'titre' qui vaut 'Biblio'.

Utilisation des paramètres dans une feuille de style

```
1  - <?xml version="1.0" encoding="iso-8859-1" ?>
2  - <xsl:stylesheet version="1.0"
3  - xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  -
5  - <!-- définition du paramètre -->
6  - <xsl:param name="titre"/>
7  -
8  - <xsl:template match="/">
9  - <html>
10 - <head>
11 - <!-- utilisation du paramètre -->
12 - <title><xsl:value-of select="$titre"/></title>
13 - </head>
14 - <body>
15 - ...
16 - ...
17 - ...
18 - ...
19 - </body>
20 - </html>
21 - </xsl:template>
22 -
23 - </xsl:stylesheet>
```

Le paramètre doit d'abord être défini avant de pouvoir être utilisé dans un template. La définition du paramètre doit préciser le nom du paramètre tel qu'il a été utilisé lors de l'appel de la feuille de style à partir de PHP.

La même façon de procéder pour le passage des paramètres est également valable dans le cas d'un recours à d'autres parseurs comme 'Saxon', 'MSXSL', 'Xalan'.

Transformation XSLT

Extension XSLT Sablotron (PHP4)

Depuis la version 4.0.6, PHP intègre une interface vers les fonctionnalités XSLT les plus courantes. Cette interface joue le rôle de couche d'abstraction entre le langage de programmation et le processeur XSLT, donnant ainsi au développeur la possibilité de passer d'un processeur à l'autre de façon transparente sans boulverser son code.

Dans les faits et pour l'instant, PHP ne supporte directement qu'un seul moteur XSLT: Sablotron. Celui-ci est chargé comme une extension et est accessible via différentes fonctions prédéfinies. PHP prévoit de prendre en charge dans un avenir "proche" d'autres processeurs, notamment Xalan.

(Remarque : l'extension XSLT a été retirée de la version 5. Pour bénéficier du support XSLT avec PHP 5, il faut utiliser l'extension XSL.)

Voici la syntaxe des deux fonctions principales permettant de lancer une transformation XSLT:

Syntaxe : `xslt_create()`

Syntaxe : `xslt_process(xslt_data)`

La fonction 'xslt_create()' initialise un processeur XSLT et 'xslt_process()' lance la transformation. 'xslt_process()' peut recevoir plusieurs paramètres, classiquement le handle du processeur XSLT, le chemin d'accès du fichier XML et le chemin d'accès du fichier XSL.

Transformation de base

Il s'agit ici d'appliquer simplement une feuille de style XSL à un fichier XML.

Transformation à partir des extensions XSLT

```
1  - <?php
2  - $xmlfile="file:///data/xml/biblio.xml";
3  - $xslfile="file:///data/xsl/biblio-sablot.xsl";
4  - $processor=xslt_create();
5  - $result=xslt_process($processor, $xmlfile, $xslfile);
6  - echo $result;
7  - ?>
```

'xslt_process' lit le fichier XML et lui applique une feuille de style XSLT. Le résultat de la transformation est stocké dans la variable $result. Pour terminer, l'instruction 'echo' envoi l'output de la transformation vers la sortie standard.
Sous Windows, il faut préfixer les chemins d'accès des fichiers que l'on veut traiter avec la suite de caractères 'file://'.

Transformation avec passage de paramètres

La fonction 'xslt_process' peut prendre plusieurs paramètres supplémentaires. Ainsi, il est possible de

passer de l'information, à partir du script PHP, à la feuille de style XSLT.

Syntaxe : `xslt_process (xslt_handle, xmlcontainer, xslcontainer [, resultcontainer [, array_arguments [, array_parameters]]])`

Le 4e argument de la fonction 'xslt_process' sert à récupérer dans un fichier le résultat de la transformation. Si il est ignoré ou fixé à 'NULL', le résultat est directement retourné.

Le 5e argument peut être utilisé afin d'éviter de devoir passer par des fichiers XML et XSLT. Les données XML et XSLT sont alors traitées directement via des variables et des tableaux.

Enfin, le 6e argument, celui qui nous intéresse tout particulièrement dans cette partie du cours, stocke le tableau associatif des données supplémentaires que l'on souhaite faire passer à la feuille de style.

Transformation avec passage de paramètres

```
1  - <?php
2  - $xmlfile = "file:///data/xml/biblio.xml";
3  - $xslfile = "file:///data/xsl/biblio-sablot.xsl";
4  - $params = array("titre" => "Bibliographie");
5  - $args = array();
6  - $prc = xslt_create();
7  - $result = xslt_process($prc, $xmlfile, $xslfile, NULL, $args,
   $params);
8  - echo $result;
9  - ?>
```

Le tableau '$params' contient le paramètre intitulé 'titre'. Il est passé en 6e argument à la fonction 'xslt_process'.

Utilisation des paramètres dans une feuille de style

```
1  - <?xml version="1.0" encoding="iso-8859-1" ?>
2  - <xsl:stylesheet version="1.0"
3  - xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  -
5  - <!-- définition du paramètre -->
6  - <xsl:param name="titre"/>
7  -
8  - <xsl:template match="/">
9  - <html>
10 - <head>
11 - <!-- utilisation du paramètre -->
12 - <title><xsl:value-of select="$title"/></title>
13 - </head>
14 - <body>
15 - ...
16 - </body>
17 - </html>
18 - </xsl:template>
19 -
20 - </xsl:stylesheet>
```

Le paramètre doit d'abord être défini avant de pouvoir être utilisé dans un template. La définition du paramètre doit préciser le nom du paramètre tel qu'il a été utilisé lors de l'appel de la feuille de style à partir de PHP.

Transformation sur base de variables

Dans les deux exemples exposés ci-dessus, les transformations se font à partir de fichiers sur disque contenant les informations xml et xsl. Il arrive toutefois dans de nombreux cas que l'information ne soit pas disponible sous forme de fichiers mais bien de variables (données xml sorties d'une DB, code xslt généré dynamiquement). Pour traiter ces flux de données, on va utiliser les buffers d'arguments de XSLT. Ils seront passés à la fonction xslt_process sous forme de tableau via le 5e paramètre.

Utilisation d'arguments comme source

```
1  - <?php
2  - $xml = "...données xml ...";
3  - $xxl = "...données xsl ...";
4  -
5  - $args = array (
6  - '/_xml' => $xml,
7  - '/_xsl' => $xsl
8  - );
9  -
10 - echo xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, $args);
11 -
12 - ?>
```

Le arguments xml et xslt sont stockés dans un tableau associatif et ensuite passés à l'instruction 'xslt_process'.

Construction d'une classe dédiée

```
1  - <?php
2  - class xsl_transform
3  - {
4  - var $args;
5  -
6  - function xsl_transform($xsl_file = '', $xml_file = '')
7  - {
8  - $xml_string = $this->read_file($xml_file);
9  - $xsl_string = $this->read_file($xsl_file);
10 - $this->args = array (
11 - '/_xml' => $xml_string ,
12 - '/_xsl' => $xsl_string
13 - );
14 - }
15 -
16 - function read_file($filename)
17 - {
18 - $fp = fopen($filename, "rb" );
19 - $content = fread($fp, filesize($filename));
20 - fclose($fp);
21 - return $content;
22 - }
23 -
24 - function apply()
25 - {
26 - $xsltproc = xslt_create();
27 - $result = xslt_process($xsltproc, 'arg:/_xml', 'arg:/_xsl', NULL,
   $this->args);
28 - return $result;
29 - }
30 - }
31 -
32 -
33 - $xslt=new xsl_transform("style.xsl","annuaire.xml");
34 - print ($xslt->apply());
35 - ?>
```

Transformation XSLT

Extension DOM XML (PHP4)

A partir de l'implémentation DOM XML par PHP, il est également possible de tirer parti des feuilles de styles XSLT. Attention, pour pouvoir exploiter cet aspect de DOM, vous devez vous assurez que les librairies 'libxslt' et 'libexslt' sont bien présentes sur votre système.

Voici les instructions à utiliser afin de lancer une transformation XSLT:

Syntaxe : `domxml_xslt_stylesheet_file(fichier_xslt)`

La fonction 'domxml_xslt_stylesheet_file()' crée un objet DomXsltStylesheet à partir d'un fichier contenant une feuille de styles XSLT.

Syntaxe : `domxml_open_file(fichier_xml)`

La fonction 'domxml_open_file()' crée un objet DomDocument à partir d'un fichier XML.

Syntaxe : `process(DomDocument, paramètres)`

La méthode 'process()' applique une transformation XSLT à un objet DomDocument.

Syntaxe : `result_dump_mem(DomDocument)`

La méthode 'result_dump_mem()' retourne le résultat d'une transfomation XSLT sous la forme d'une chaîne de caractères.

Transformation de base

Il s'agit ici d'appliquer simplement une feuille de styles XSLT à un fichier XML.

Transformation à partir de DOM XML

```
1  - <?php
2  - $fichierxml = "D:\xml\bilio.xml";
3  - $fichierxsl = "D:\xsl\bilio.xsl";
4  - $xslt = domxml_xslt_stylesheet_file($fichierxsl);
5  - $xml = domxml_open_file($fichierxml);
6  - $result = $xslt->process($xml);
7  - print $xslt->result_dump_mem($result);
8  - ?>
```

La fonction 'domxml_xslt_stylesheet_file()' crée un objet DomXsltStylesheet qui est représenté par la variable $xslt.
La fonction 'domxml_open_file()' crée juste après un objet DomDocument à partir du fichier XML "biblio.xml".
Ensuite, la transformation est appliquée via la méthode "process" de l'objet DomXsltStylesheet ($xslt).
Pour terminer, le résultat de la transformation est envoyé sur la sortie standard par l'intermédiare de la méthode "result_dump_mem()".

Transformation avec passage de paramètres

La méthode 'process' de l'objet DomXsltStylesheet peut recevoir en second argument un tableau de paramètres. Celui-ci sera passé à la feuille de styles XSLT.

Syntaxe : `process(DomDocument [, array_parameters])`

Le deuxième argument stocke le tableau associatif des données supplémentaires que l'on souhaite faire passer à la feuille de styles.

Transformation avec passage de paramètres

```
1  - <?php
2  - $fichierxml = "D:\xml\bilio.xml";
3  - $fichierxsl = "D:\xsl\bilio.xsl";
4  - $xslt = domxml_xslt_stylesheet_file($fichierxsl);
5  - $xml = domxml_open_file($fichierxml);
6  - $params = array("titre" => "Bibliographie");
7  - $result = $xslt->process($xml, $params);
8  - print $xslt->result_dump_mem($result);
9  - ?>
```

Le tableau '$params' contient le paramètre intitulé 'titre'. Il est passé en second argument à la méthode 'process'.

Utilisation des paramètres dans une feuille de styles

```
1  - <?xml version="1.0" encoding="iso-8859-1" ?>
2  - <xsl:stylesheet version="1.0"
3  - xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  -
5  - <!-- définition du paramètre -->
6  - <xsl:param name="titre"/>
7  -
8  - <xsl:template match="/">
9  - <html>
10 - <head>
11 - <!-- utilisation du paramètre -->
12 - <title><xsl:value-of select="$title"/></title>
13 - </head>
14 - <body>
15 - ...
16 - </body>
17 - </html>
18 - </xsl:template>
19 -
20 - </xsl:stylesheet>
```

Le paramètre doit d'abord être défini avant de pouvoir être utilisé dans un template. La définition du paramètre doit préciser le nom du paramètre tel qu'il a été utilisé lors de l'appel de la feuille de style à partir de PHP.

Transformation XSLT

Extension XSL (PHP5)

A partir de la version 5 de PHP, c'est l'extension XSL qui implémente le standard XSL. Elle effectue les transformations XSLT à l'aide de la bibliothèque libxslt (réputée très rapide et supportant plusieurs extensions exslt).

Exemples d'utilisation

Transformation de base

```
1  - <?php
2  - $xml = new DomDocument;
3  - $xml->load('data.xml');
4  -
5  - $xsl = new DomDocument;
6  - $xsl->load('style.xsl');
7  -
8  - $proc = new XSLTProcessor();
9  - $proc->importStyleSheet($xsl);
10 - echo $proc->transformToXML($xml);
11 - ?>
```

Les grandes étapes de la transformation :
1) chargement du code XML grâce à la classe domDocument et à sa méthode 'domDocument::load()' qui charge un fichier à partir d'une URL. Le chargement du code XML peut également se faire à partir d'une chaîne de caractères; dans ce cas il faut utiliser la méthode 'domDocument::loadXML()';
2) chargement du fichier XSL selon la méthode décrite ci-dessus;
3) nouvelle instance de la classe XSLTProcessor;
4) importation de l'objet représentant la feuille de style xslt dans le processeur XSLT (emploi de la méthode 'XSLTProcessor::importStylesheet()');
5) lancement de la transformation proprement dite via la méthode 'XSLTProcessor::TransformToXml()'.

Transformation avec passage de paramètres

```
1  - <?php
2  - $xml = new DomDocument;
3  - $xml->load('data.xml');
4  -
5  - $xsl = new DomDocument;
6  - $xsl->load('style.xsl');
7  -
8  - $proc = new XSLTProcessor();
9  - $proc->setParameter("", "title", "Biblio");
10 - $proc->setParameter("", "date", Date("Ymd"));
11 -
12 - $proc->importStyleSheet($xsl);
13 - echo $proc->transformToXML($xml);
14 - ?>
```

Les paramètres sont fixés via la méthode 'XSLTProcessor::setParameter()'.

Passage de paramètres multiples

```
1  - <?php
2  - $xml = new DomDocument;
3  - $xml->load('data.xml');
4  -
5  - $xsl = new DomDocument;
6  - $xsl->load('style.xsl');
7  -
8  - $proc = new XSLTProcessor();
9  -
10 - foreach ($_GET as $key => $value)
11 -    {
12 -    $proc->setParameter("", $key, $value);
13 -    }
14 -
15 - $proc->importStyleSheet($xsl);
16 - echo $proc->transformToXML($xml);
17 - ?>
```

Les paramètres envoyés au script PHP par la méthode GET sont tous passés à la feuille de style XSLT. Les paramètres qui ne sont pas reconnus par celle-ci seront simplement ignorés lors de la transformation.
A partir de la version 5.1 de PHP, la méthode setParameter peut également être invoquée de façon à prendre en compte non plus un seul paramètre à la fois, mais bien un tableau (array) de paramètres : setParameter (string namespace, array parameters)

Transformation et enregistrement du résultat dans un fichier

```
1  - <?php
2  - $xml = new DomDocument;
3  - $xml->load('data.xml');
4  -
5  - $xsl = new DomDocument;
6  - $xsl->load('style.xsl');
7  -
8  - $proc = new XSLTProcessor();
9  - $proc->importStyleSheet($xsl);
10 - $proc->transformToUri($xml, 'dump.xml');
11 - ?>
```

L'utilisation de la méthode 'XSLTProcessor::transformToUri()' a pour effet de créer sur disque le fichier 'dump.xml' qui contiendra tous les codes générés par la feuille de style xslt.

Manipulation DOM

Modèle et mode de fonctionnement DOM - Modélisation DOM

Le Document Object Model (DOM) est une norme définie par le W3C. Elle offre une interface standard pour accéder et manipuler des ensembles structurés de données. Appliquée au format XML, elle modélise un document (ou une chaîne de caractères) sous forme de structure arborescente. Cette structure est composée hiérarchiquement d'objets. Ces objets présentent des attributs et des méthodes qui peuvent être exploités par une application afin de manipuler un document XML.

Concrètement, DOM, lorsqu'elle est sollicitée, charge une chaîne de caractères ou un fichier XML en mémoire, en analyse la structure et permet aux développeurs de traiter le tout comme un ensemble cohérent.

Modélisation DOM

Exemple de document XML:

```
<?xml version="1.0"?>
<auteurs>
  <nom>James Ellroy</nom>
  <nom>Jack London</nom>
  <nom>Jules Vernes</nom>
  <nom>Umberto Eco</nom>
  <nom>Dan Simmons</nom>
</auteurs>
```

Modèle DOM :

```
DomDocument
    |
    |--DomNode 'auteurs'
            |
            |-->DomNode 'nom'
            |       |
            |       |-->DomNode text 'James Ellroy'
            |
            |-->DomNode 'nom'
            |       |
            |       |-->DomNode text 'Jack London'
            |
            |-->DomNode 'nom'
            |       |
            |       |-->DomNode text 'Jules Vernes'
            |
            |-->DomNode 'nom'
```

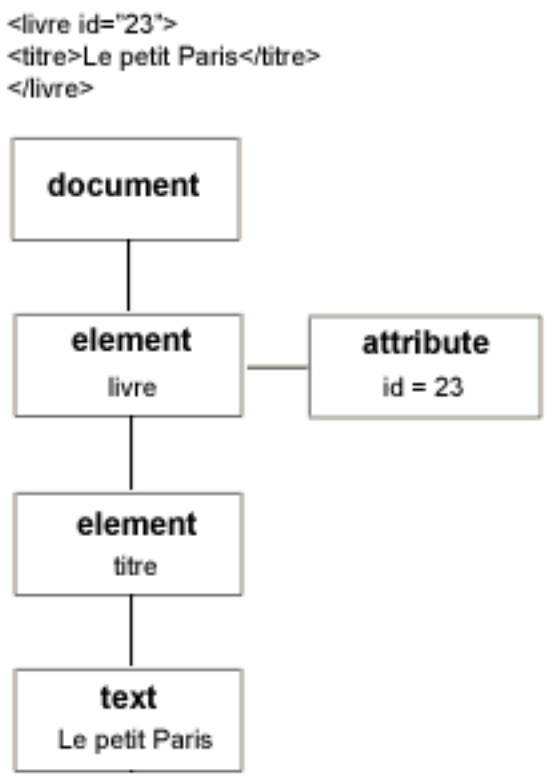
```
|          |
|          |-->DomNode text 'Umberto Eco'
|
|-->DomNode 'nom'
|          |
|          |-->DomNode text 'Dan Simmons'
```

On le voit dans cet exemple, la structure XML est bien comprise par DOM comme étant composée d'un noeud racine 'auteurs', contenant des noeuds enfants 'nom' qui eux-mêmes encapsulent des noeuds de type texte.

DOM, dans le cadre de la représentation d'un document XML, définit plusieurs objets qui vont servir de points d'entrée pour la manipulation des données. Ils représentent chacun un des composants types de l'arborescence du document :

- DomNode Représente un noeud. C'est le type d'objet principal (les autres en sont dérivés).
- DomDocument Représente le noeud document ('/').
- DomElement Représente un noeud 'élément'.
- DomAttribute Représente un noeud 'attribut'.

Représentation XML et objets DOM



Ces objets s'accompagnent de différentes propriétés et méthodes (définies par le W3C) qui rendent possible la gestion de l'ensemble du document : création de documents, création de noeuds, ajout d'attributs, modification de noeuds textes...

Attention cependant : il faut être conscient que l'API DOM présente l'inconvénient qu'elle impose de construire un arbre en mémoire contenant l'intégralité des éléments du document. Pour des documents dont la taille serait fort conséquente, cette façon de procéder peut se révéler handicapante.

Manipulation DOM

Modèle et mode de fonctionnement DOM - Extension DOM XML

PHP dans sa version 4, à travers sa librairie 'libxml' et son extension DOM XML, supporte le modèle DOM. Voici les propriétés et les méthodes associées aux principaux objets DOM.

La création d'un objet DomDocument vide peut être obtenue via la fonction 'domxml_new_doc'.

Syntaxe : `domxml_new_doc(version)`

Cette fonction constitue la plupart du temps un passage obligé afin de travailler avec DOM. Si l'on désire travailler à partir d'un document XML existant, on utilisera la fonction 'domxml_open_file()'.

Syntaxe : `domxml_open_file(file_path)`

Une fois l'objet document obtenu, il est possible d'exploiter ses propriétés et méthodes. C'est également à partir de cet objet que pourront être créés les objets DomElement, DomNode et DomAttribute. Voyons cela de plus près.

Classe : `DomDocument`

Propriétés de DomDocument

`version`
version xml du document
ex : echo $doc->version;

`encoding`
type d'encodage pour le texte (jeu de caractères)
ex : echo $doc->encoding;

`standalone`
autonomie du document
ex : echo $doc->standalone;

Méthodes de DomDocument

`document_element()`
retourne l'élément racine
ex : $root = $doc->document_element();

`create_comment(comment_text)`
crée un nouveau commentaire
ex : $comment = $doc->create_comment("DOM/PHP");

`create_element(element_name)`
crée un nouvel élément
ex : $livres = $doc->create_element("livres");

`create_text_node(text)`

crée un nouveau noeud texte
ex : $text = $doc->create_text_node("yyy")

Classe : `DomNode`

Méthodes de DomNode

`child_nodes()`
retourne les noeuds enfants (sous forme d'un tableau de noeuds)
ex : $children = $root->child_nodes();

`first_child()`
retourne le premier noeud enfant
ex : $fnode = $biblio->first_child();

`last_child()`
retourne le dernier noeud
ex : $lnode = $livres->last_child();

`parent_node()`
retourne le noeud parent
ex : $parentnode = $cur_node->parent_node();

`node_name()`
retourne le nom d'un noeud
ex : if ($cur_node->node_name() == "titre")

`node_type()`
retourne le type d'un noeud
ex : if ($cur_node->node_type() == XML_ELEMENT_NODE)

`get_content()`
retourne le contenu d'un noeud
ex : $str = $cur_node->get_content();

`append_child(object newnode)`
ajoute un noeud enfant
ex : $livres->append_child($livre);

`remove_child(object child_node)`
supprime un noeud d'une liste de noeuds enfants
ex : $rnode = $livres->remove_child($children[0]);

Classe : `DomElement (étend DomNode)`

Méthodes de DomElement

`tagname()`
retourne le nom de l'élément
ex : echo $name->tagname();

`get_elements_by_tagname(tag_name)`
récupération de noeuds sur base de leur nom (sous forme d'un tableau de noeuds)

ex : $nodelist = $root->get_elements_by_tagname("livre");

`set_attribute(name, value)`
ajoute un nouvel attribut à un noeud
ex : $bibliographie->set_attribute("auteur", "Circum Net");

`has_attribute(attribute_name)`
teste si un attribut existe
ex : if ($livre->has_attribute("titre"))

`get_attribute(attribute_name)`
retourne un noeud attribut
ex : $titre = $livre->get_attribute("titre");

Dans les articles suivants, nous verrons comment -concrètement- mettre en oeuvre ses instructions et en tirer profit.

Conversion des jeux de caractères

La plupart des modules de manipulation XML traitent les données qu'ils fournissent et qu'ils réceptionnent selon le codage international UTF-8.
Si des caractères accentués ou spéciaux doivent être utilisés (codage ISO-8859-1), il faudra avoir recours aux fonctions PHP 'utf8_encode' et 'utf8_decode' pour que la conversion des chaînes de caractères se passe correctement.

- utf8_encode fait passer une chaîne d'iso-8859-1 à utf-8
- utf8_decode fait passer une chaîne de utf-8 à iso-8859-1

Protection des caractères réservés

Si l'on veut utiliser en tant que données, dans un document XML, des caractères ayant une signification particulière (comme <, >, et &) sans qu'ils soient interprétés, il faut passer par le biais des "entités". Une entité est une suite de caractères commençant par & et se terminant par ; (par exemple l'entité pour le signe < est <).

En PHP, la fonction 'htmlspacialchars()' convertit automatiquement tous les caractères spéciaux en entités HTML/XML. Voici les remplacements qui sont effectués :

- "&" (et commercial) devient "&"
- """ (guillemets doubles) devient """
- "<" (supérieur à) devient "<"
- ">" (inférieur à) devient ">"

Manipulation DOM

Modèle et mode de fonctionnement DOM - Extension DOM

A partir de PHP 5, c'est l'extension DOM qui permet de manipuler des documents XML avec l'API DOM. Elle remplace l'extension DOM XML disponible dans la version 4.

Ce module s'appuie sur les standards niveau 2 de DOM et est donc clairement orienté objet. En voici les principaux composants :

Classe : `DOMDocument`

Propriétés de DOMDocument

`version`
version xml du document

`encoding`
type d'encodage pour le texte (jeu de caractères)

`documentElement`
accès direct à l'élément racine d'un document

Méthodes de DOMDocument

`load()`
charge un contenu XML à partir d'un fichier
exemple :
$doc = new DOMDocument();
$doc->load('livres.xml');

`loadXML(source)`
charge un contenu XML à partir d'une chaîne de caractères
exemple :
$doc = new DOMDocument();
$doc->loadXML('<biblio><livre>...</livre></biblio>');

`save(filename)`
sauvegarde une structure XML dans un fichier
exemple :
$doc->load('data/biblio.xml');

`saveXML()`
sauvegarde une structure XML dans une chaîne de caractères
exemple :
echo $doc->saveXML() . "\n";
dump du document entier
echo $doc->saveXML($author) . "\n";
dump d'un noeud particulier

`createElement(name [, value])`

crée un nouvel élément
exemple :
$element = $doc->createElement('livre', 'test');

`createTextNode(text)`
crée un nouveau noeud texte (retourne un objet DOMText)
exemple :
$dom->createTextNode("Le petit prince")

`getElementsByTagName(tag_name)`
récupération de noeuds sur base de leur nom (retourne un objet DOMNodeList)
exemple :
$livres = $dom->getElementsByTagName('livre');

`validate()`
valide un document en se basant sur sa DTD.

Classe : `DomNode`

Propriétés de DOMNode

`childNodes`
retourne les noeuds enfants (retourne un objet DOMNodeList)
exemple :
$root = $doc->documentElement;
$children = $root->childNodes;

`firstChild`
retourne le premier noeud enfant (retourne un objet DOMNode)

`lastChild`
retourne le dernier noeud (retourne un objet DOMNode)

`parentNode`
retourne le noeud parent (retourne un objet DOMNode)

`nodeName`
retourne le nom d'un noeud
exemple :
if ($curNode->nodeName == "titre")

`nodeType`
retourne le type d'un noeud
exemple :
if ($curNode->nodeType == XML_ELEMENT_NODE)

`nodeValue`
retourne la valeur d'un noeud (en fonction de son type)
exemple :
$content = trim($element->nodeValue);

Méthodes de DomNode

`hasAttributes()`
vérifie si un noeud possède des attributs
exemple :
if (!$livre-->hasAttributes())

`hasChildNodes()`
vérifie si un noeud possède des enfants
exemple :
if (!$livre-->hasChildNodes())

`appendChild(object newnode)`
ajoute un noeud enfant
exemple :
$livres-->appendChild($livre);

`removeChild(object child_node)`
supprime un noeud d'une liste de noeuds enfants
exemple :
$rnode = $livre-->removeChild($prix);

Classe : `DOMElement (étend DOMNode)`

Propriétés de DOMNode

`tagName`
le nom de l'élément
exemple :
if ($childNode-->tagName == 'livre')

Méthodes de DomElement

`getElementsByTagName(tag_name)`
récupération de noeuds sur base de leur nom (retourne un objet DOMNodeList)
exemple :
$auteurs = $livre->getElementsByTagName('auteur');

`setAttribute(name, value)`
ajoute un nouvel attribut à un noeud
exemple :
$bibliographie->setAttribute("auteur", "Circum Net");

`hasAttribute(attribute_name)`
teste si un attribut existe
exemple :
if ($livre->hasAttribute("id"))

`getAttribute(attribute_name)`
récupère la valeur d'un attribut
exemple :
$id = $livre->getAttribute("id");

Manipulation DOM

Opérations DOM XML - Module DOM XML PHP4 - Construction d'un arbre

Il est possible, à partir d'un script PHP, de construire de bout en bout un document XML en exploitant le modèle DOM.

L'utilisation de l'interface DOM présente ici l'avantage d'une couche d'abstraction supplémentaire. L'écriture des balises n'étant plus du ressort direct du programmeur, les erreurs de frappe et de structure sont évitées, la portabilité est renforcée et la standardisation est respectée.

Passons en revue les instructions PHP qui vont être utilisées :

- domxml_new_doc(version) création d'un document DOM vide / obtention d'un objet DomDocument
- create_element(node_name) création d'un noeud élément / obtention d'un objet DomElement
- create_text_node(text) création d'un noeud texte
- set_attribute(name, value) ajout d'un attribut
- append_child(new_node) ajout d'un enfant à un noeud

Pour construire un document XML selon DOM, il faut commencer par créer un objet DomDocument (avec 'domxml_new_doc'), lui ajouter un noeud racine (avec 'create_element()'), et ensuite définir tous les noeuds enfants dont il sera composé (avec 'create_element()' et 'append_child'). L'exemple ci-dessous illustre cette procédure de création.

Construction d'un arbre DOM

```
1  - <?php
2  - # Création d'un objet Document
3  - $doc = domxml_new_doc("1.0");
4  -
5  - # Création d'un commentaire
6  - $comment = $doc->create_comment("Test DOM/PHP");
7  - $doc->append_child($comment);
8  -
9  - # Création d'un élément racine
10 - $bibliographie = $doc->create_element("bibliographie");
11 - $bibliographie->set_attribute("auteur", "Circum Net");
12 - $doc->append_child($bibliographie);
13 -
14 - # Ajout de l'élément 'livres' à la racine
15 - $livres = $doc->create_element("livres");
16 - $bibliographie->append_child($livres);
17 -
18 - # Ajout d'un noeud "enfant" à l'élément 'livres'
19 - $livre = $doc->create_element("livre");
20 - $titre = $doc->create_element("titre");
21 - $titre->append_child($doc->create_text_node("xxx"));
22 - $livre->append_child($titre);
23 - $livres->append_child($livre);
24 -
25 - # Ajout d'un noeud "enfant" à l'élément 'livres'
26 - $livre = $doc->create_element("livre");
27 - $titre = $doc->create_element("titre");
28 - $titre->append_child($doc->create_text_node("yyy"));
29 - $livre->append_child($titre);
30 - $livres->append_child($livre);
31 -
32 - # Ajout d'un noeud "enfant" à l'élément 'livres'
33 - $livre = $doc->create_element("livre");
34 - $titre = $doc->create_element("titre");
35 - $titre->append_child($doc->create_text_node("zzz"));
36 - $livre->append_child($titre);
37 - $livres->append_child($livre);
38 -
```

```
39 - # Récupération de l'arbre DOM
40 - # sous forme d'une chaîne de caractères
41 - echo $doc->dump_mem(true);
42 - ?>
```

Dans cet exemple, le résultat de la transformation est redirigé vers la sortie standard. Pour enregistrer le résultat dans un fichier, il suffit d'écrire la chaîne de caractères obtenue via 'dump_mem()' en utilisant la fonction 'fwrite()'.

Dans la plupart des cas, les informations à stocker dans un fichier XML ne seront pas "hardcodées" dans le script, mais viendront plus probablement d'une autre source de données : un fichier csv, un fichier xml, une base de données. Voyons comment procéder en supposant que les informations proviennent de la consultation d'une base de données MySQL.

Construction d'un arbre DOM à partir de MySQL

```
1  - <?php
2  - # Création d'un objet Document
3  - $doc = domxml_new_doc("1.0");
4  -
5  - # Création d'un commentaire
6  - $comment = $doc->create_comment("Test DOM/PHP");
7  - $doc->append_child($comment);
8  -
9  - # Création d'un élément racine
10 - $bibliographie = $doc->create_element("bibliographie");
11 - $bibliographie->set_attribute("auteur", "Circum Net");
12 - $doc->append_child($bibliographie);
13 -
14 - # Ajout de l'élément 'livres' à la racine
15 - $livres = $doc->create_element("livres");
16 - $bibliographie->append_child($livres);
17 -
18 - $link = mysql_connect("localhost", "user", "pass");
19 - $bdlink = mysql_select_db("biblio", $link);
20 - $result = mysql_query ("SELECT * FROM livres", $link);
21 - while($myrow=mysql_fetch_array($result))
22 - {
23 - # Ajout d'un noeud "enfant" à l'élément livres
24 - $livre = $doc->create_element("livre");
25 - $titre = $doc->create_element("titre");
26 - $titre->append_child($doc->create_text_node($myrow[titre]));
27 - $livre->append_child($titre);
28 - $livres->append_child($livre);
29 - }
30 -
31 - # Récupération de l'arbre DOM
32 - # sous forme d'une chaîne de caractères
33 - echo $doc->dump_mem(true);
34 - ?>
```

Les données sont ici récupérées à partir d'une base de données et via une requête SQL pour être intégrées dans un document XML.

Manipulation DOM

Opérations DOM XML - Module DOM XML PHP4 - Ajout d'un noeud

L'ajout d'un noeud dans un document xml existant va se faire en deux étapes : dans un premier temps on charge en mémoire le document XML sous forme d'arbre DOM et ensuite on ajoute à l'endroit voulu le nouvel élément.

Voici les principales instructions exploitées pour cette manipulation :

- domxml_open_file(filename) obtention d'un objet DomDocument à partir d'un fichier
- document_element() récupération du noeud racine
- get_elements_by_tagname(nodename) récupération de noeuds sur base de leur nom
- append_child(new_node) ajout d'un enfant à un noeud

Procédure d'ajout d'un noeud dans un arbre DOM

```
1  - <?php
2  - function addBook($title, $author, $editor)
3  - {
4  - global $doc, $root;
5  - $elementBook = $doc->create_element("livre");
6  - $elementTitle = $doc->create_element("titre");
7  - $elementTitle->append_child($doc->create_text_node($title));
8  - $elementBook->append_child($elementTitle);
9  - $elementAutho = $doc->create_element("auteur");
10 - $elementAutho->append_child($doc->create_text_node($author));
11 - $elementBook->append_child($elementAutho);
12 - $elementEditor = $doc->create_element("editeur");
13 - $elementEditor->append_child($doc->create_text_node($editor));
14 - $elementBook->append_child($elementEditor);
15 -
16 - $books = $root->get_elements_by_tagname("livres");
17 - $books[0]->append_child($elementBook);
18 - }
19 -
20 - $xml_file = realpath("biblio.xml");
21 - $doc = domxml_open_file($xml_file);
22 - $root = $doc->document_element();
23 -
24 - addBook("Ma part d'ombre", "James Ellroy", "Rivages");
25 -
26 - echo $doc->dump_mem(true);
27 - ?>
```

Manipulation DOM

Opérations DOM XML - Module DOM XML PHP4 - Parcours d'un arbre

Dans le chapitre précédent, nous avons vu comment créer un document XML de toute pièce. Ici, il s'agit d'exploiter les fonctions DOM XML de PHP afin de parcourir tous les éléments dont est constitué un document XML existant, d'en repérer certaines parties, d'en modifier ou d'en supprimer d'autres.

Sélection de noeuds

Pour illustrer la procédure de sélection d'un noeud dans un document XML, nous allons réaliser un petit outil de recherche à partir des instructions DOM. Voici les fonctions prédéfinies qui vont être sollicitées :

- domxml_open_file() : création d'un document DOM à partir d'un fichier XML
- document_element() : récupération du noeud racine
- get_elements_by_tagname() : récupération de noeuds sur base de leur nom
- get_content() : récupération du contenu d'un noeud

Imaginons un formulaire HTML qui récupère un mot-clé en vue d'effectuer une recherche dans une bibliographie au format XML. A quoi devrait ressembler un script PHP chargé d'accomplir cette tâche. Examinons le script ci-dessous.

Outil de recherche DOM

```
1  - <?php
2  - # mot-clé récupéré à partir du formulaire
3  - $keyword = "London";
4  -
5  - # fichier xml
6  - $xml_file = "D:\xml\biblio.xml";
7  -
8  - # création d'un objet DomDocument
9  - if(!$doc = domxml_open_file($xml_file))
10 - {
11 - die("Le document XML contient des erreurs !");
12 - }
13 -
14 - # récupération de la racine et des noeuds enfants "livre"
15 - $root = $doc->document_element();
16 - $nodelist = $root->get_elements_by_tagname("livre");
17 -
18 - for ($i=0; $i<sizeof($nodelist); $i++)
19 - {
20 - $titles=$nodelist[$i]->get_elements_by_tagname("titre");
21 - if(eregi($keyword,$titles[0]->get_content()))
22 -    {
23 -    $title_str=$titles[0]->get_content();
24 -    echo "<p>" . utf8_decode($title_str) . "</p>";
25 -    }
26 - }
27 - ?>
```

Ce script commence par créer un objet DOM sur base du document XML fourni. Ensuite, il récupère tous les noeuds "livre" à partir de la racine de l'arbre DOM.
Les noeuds livres sont stockés dans un tableau scalaire. Pour chaque noeud de ce tableau, le script récupère l'élément "titre" et teste s'il contient le critère de recherche ($keyword). Dans l'affirmative, le titre est affiché à l'écran.

Parcours d'arbre

Voyons à présent comment, en suivant le modèle DOM, il est possible de parcourir l'ensemble des noeuds composant un document XML. Voici les fonctions prédéfinies qui vont être utilisées :

- domxml_open_file() : création d'un document DOM à partir d'un fichier XML
- document_element() : récupération du noeud racine
- child_nodes() : retourne un tableau contenant les noeuds enfants
- node_type() : récupération du type d'un noeud
- get_content() : récupération du contenu d'un noeud

Imaginons un document XML dont nous voudrions passer en revue tous les noeuds texte afin d'y repérer une chaîne de caractères. Voyons quelle serait la procédure à suivre :

Parcours d'un arbre DOM

```
1  - <?php
2  - # critère de recherche
3  - $keyword = "London";
4  -
5  - # création d'un objet DOM à partir du document XML
6  - $iml_file = "D:\xml\biblio.xml";
7  - if(!$doc = domxml_open_file($iml_file))
8  - {
9  - die("Le document XML contient des erreurs !");
10 - }
11 -
12 - # récupération de la racine
13 - $root = $doc->document_element();
14 -
15 - # récupération des noeuds enfants
16 - $children = $root->child_nodes();
17 -
18 - # appel de la fonction qui effectue la recherche
19 - search_text($children, $keyword);
20 -
21 - # fonction (récursive) qui passe en revue tous les éléments
22 - function search_text($nodelist, $search)
23 - {
24 - for ($i=0; $i<sizeof($nodelist); $i++)
25 - {
26 - if($nodelist[$i]->node_type()==XML_ELEMENT_NODE)
27 -    {
28 -    $nextCollection=$nodelist[$i]->child_nodes();
29 -    search_text($nextCollection, $search);
30 -    }
31 - elseif($nodelist[$i]->node_type()==XML_TEXT_NODE)
32 -    {
33 -    if(eregi($search,$nodelist[$i]->get_content()))
34 -       {
35 -       $str=$nodelist[$i]->content;
36 -       echo "<p>" . utf8_decode($str) . "</p>";
37 -       }
38 -    }
39 - }
40 - }
41 - ?>
```

Ce script crée un objet DOM sur base du document XML fourni. Il récupère les noeuds enfants de la racine de l'arbre DOM. A partir de là, la fonction 'search_text' est invoquée.
Dans cet exemple, nous utilisons la récursivité: à chaque fois qu'un noeud de type 'élément' est rencontré la fonction 'search_text' est appellée (elle récupère en argument les noeuds enfants du noeud courant).
Si c'est un noeud de type 'texte' qui est traité, alors l'instruction 'eregi' vérifie si le critère de recherche ($keyword) est présent dans le contenu textuel. Dans l'affirmative, le contenu textuel est affiché.

Manipulation DOM

Opérations DOM XML - Module DOM XML PHP4 - Exploitation d'XPATH

Comme nous avons eu l'occasion de le voir dans l'introduction, XPATH est un langage d'expressions qui permet de sélectionner des éléments ou des groupes d'éléments au sein d'un document XML. PHP, à travers ses fonctions DOM XML, permet l'utilisation de ces expressions.

Voyons les instructions PHP qu'il est indispensable de connaître pour ce type de manipulation :

- xpath_new_context() création d'un contexte xpath à partir d'un objet DomDocument
- xpath_eval (xpath_context, xpath_expression) évaluation de l'expression et obtention d'un objet dont la propriété 'nodeset' référence un array contenant les noeuds trouvés

Pour illustrer l'exploitation d'XPATH via DOM XML, nous allons établir la liste des titres contenus dans une bibliographie au format XML.

Utilisation d'XPATH

```
1  - <?php
2  - # parsing du document XML
3  - $xml_file = "D:\xml\biblio.xml";
4  - if(!$doc = domxml_open_file($xml_file))
5  - {
6  - die("Le document XML contient des erreurs !");
7  - }
8  -
9  - # création d'un contexte XPATH
10 - $ctx = $doc->xpath_new_context();
11 -
12 - # expression XPATH
13 - $xpath_expr = "/bibliographie/livres/livre/titre";
14 -
15 - # récupération de noeuds avec XPATH
16 - # retourne un objet qui présente une propriété appellée nodeset
17 - # nodeset référence un array contenant les noeuds
18 - # récupérés via l'expression xpath
19 - $nodelist = $ctx->xpath_eval($xpath_expr);
20 -
21 - # teste si un résultat a été obtenu
22 - if (is_array($nodelist->nodeset))
23 -    {
24 -    # affichage du titre
25 -    foreach ($nodelist->nodeset as $node)
26 -       {
27 -       $textnode = $node->first_child();
28 -       print utf8_decode($textnode->node_value()) . "<br>";
29 -       }
30 -    }
31 - }
32 - ?>
```

Les noeuds "titre" sont récupérés grâce à l'expression "/bibliographie/livres/livre/titre". Il sont disponibles à travers la propriété "nodeset" de l'objet "$nodelist". Cette propriété donne accès à un array contenant l'ensemble des noeuds résultats.
Un test est effectué pour voir si l'expression XPATH a bien retourné un résultat : on vérifie pour cela si la propriété 'nodeset' référence bien un tableau (is_array).
En parcourant le tableau résultat on prend les noeuds textes des éléments 'titre' (via 'first_child' et 'node_value') pour les afficher à l'écran.

Manipulation DOM

Opérations DOM - Module DOM PHP5 - Construction d'un arbre

La construction d'un arbre DOM sur base de la nouvelle extension de PHP5 repose sur une procédure quasi identique à la précédente.

Voici les principales méthodes qui vont être utilisées :

- createComment(value) création d'un commentaire
- createElement(node_name [,value]) création d'un noeud élément / obtention d'un objet DomElement
- createTextNode(text) création d'un noeud texte
- setAttribute(name, value) ajout d'un attribut
- appendChild(new_node) ajout d'un enfant à un noeud
- saveXML([node]) récupère l'arbre DOM sous forme d'une chaîne

Construction d'un arbre DOM - structure simple

```
1  - <?php
2  - # Création d'une nouvelle structure DOM
3  - $doc = new DOMDocument('1.0', 'iso-8859-1');
4  -
5  - # Création et ajout d'un élément racine
6  - $element = $doc->createElement('livre', 'contenu de test');
7  - $doc->appendChild($element);
8  -
9  - # Conversion de la structure DOM en une chaîne de caractères
10 - # Affichage de la chaîne obtenue
11 - echo $doc->saveXML();
12 -
13 - # Résultat obtenu et affiché :
14 - # <?xml version="1.0" encoding="iso-8859-1"?>
15 - # <livre>contenu de test</livre>
16 - ?>
```

Construction d'un arbre DOM - structure sophistiquée

```
1  - <?php
2  - # Création d'une nouvelle structure DOM
3  - $dom = new DOMDocument('1.0', 'iso-8859-1');
4  -
5  - # Création et ajout d'un commentaire
6  - $comment = $dom->createComment("Test DOM/PHP");
7  - $dom->appendChild($comment);
8  -
9  - # Création et ajout d'un élément racine
10 - $bibliographie = $dom->createElement("bibliographie");
11 - $bibliographie->setAttribute("auteur", "Circum Net");
12 - $dom->appendChild($bibliographie);
13 -
14 - $livres = $dom->createElement("livres");
15 - $bibliographie->appendChild($livres);
16 -
17 - $livre = $dom->createElement("livre");
18 - $titre = $dom->createElement("titre");
19 - $titre->appendChild($dom->createTextNode("Clandestin"));
20 - $livre->appendChild($titre);
21 - $auteur = $dom->createElement("auteur");
22 - $auteur->appendChild($dom->createTextNode("James Ellroy"));
23 - $livre->appendChild($auteur);
24 - $editeur = $dom->createElement("editeur");
25 - $editeur->appendChild($dom->createTextNode("Rivages"));
26 - $livre->appendChild($editeur);
```

```
27 - $livres->appendChild($livre);
28 -
29 - $livre = $dom->createElement("livre");
30 - $titre = $dom->createElement("titre");
31 - $titre->appendChild($dom->createTextNode("Le Petit Prince"));
32 - $livre->appendChild($titre);
33 - $auteur = $dom->createElement("auteur");
34 - $auteur->appendChild($dom->createTextNode(""));
35 - $livre->appendChild($auteur);
36 - $editeur = $dom->createElement("editeur");
37 - $editeur->appendChild($dom->createTextNode("Gallimard"));
38 - $livre->appendChild($editeur);
39 - $livres->appendChild($livre);
40 -
41 - $livre = $dom->createElement("livre");
42 - $titre = $dom->createElement("titre");
43 - $titre->appendChild($dom->createTextNode("Barberousse"));
44 - $livre->appendChild($titre);
45 - $auteur = $dom->createElement("auteur");
46 - $auteur->appendChild($dom->createTextNode("Michel Tournier"));
47 - $livre->appendChild($auteur);
48 - $editeur = $dom->createElement("editeur");
49 - $editeur->appendChild($dom->createTextNode("Gallimard"));
50 - $livre->appendChild($editeur);
51 - $livres->appendChild($livre);
52 -
53 - # Conversion de la structure DOM en une chaîne de caractères
54 - # Affichage de la chaîne obtenue
55 - echo $dom->saveXML();
56 -
57 - # Sauvegarde l'arbre XML dans un fichier
58 - $dom->save("biblio.xml");
59 - ?>
```

Construction d'un arbre DOM à partir d'une requête SQL

```
1  - <?php
2  - # Connexion au serveur et sélection de la base
3  - $db = mysql_connect('localhost', 'user', 'pass');
4  - mysql_select_db('biblio');
5  -
6  - # Exécution de la requête SQL
7  - $query = "SELECT * FROM livres";
8  - $result = mysql_query($query, $db);
9  -
10 - # Création d'une nouvelle structure DOM
11 - $dom = new DOMDocument('1.0', 'iso-8859-1');
12 -
13 - # Création et ajout d'un élément racine
14 - $biblio = $dom->createElement("bibliographie");
15 - $biblio->setAttribute("auteur", "Circum Net");
16 - $dom->appendChild($biblio);
17 -
18 - # Traitement -une à une- des lignes du résultat
19 - while($row = mysql_fetch_assoc($result))
20 -    {
21 -    # Ajout d'un noeud pour chaque ligne
22 -    $book = $dom->createElement("livre");
23 -    $biblio->appendChild($book);
24 -
25 -    # Ajout d'un noeud enfant pour chaque champs
26 -    foreach ($row as $fieldname => $fieldvalue)
27 -       {
28 -       # Création et ajout du noeud enfant
29 -       $child = $dom->createElement($fieldname);
30 -       $book->appendChild($child);
31 -
32 -       # Ajout de la valeur du champs
33 -       # en tant que noeud texte
34 -       $value = $dom->createTextNode($fieldvalue);
35 -       $child->appendChild($value);
36 -       }
37 -    }
38 -
```

```
39 - # Sauvegarde l'arbre XML dans un fichier
40 - $dom->save("biblio.xml");
41 -
42 - # Fermeture de la connexion
43 - mysql_close($db);
44 - ?>
```

Manipulation DOM

Opérations DOM - Module DOM PHP5 - Ajout d'un noeud

De la même façon que pour l'extension DOM XML de PHP4, ici l'ajout d'un noeud va se faire également en deux étapes : d'abord charger en mémoire le document XML sous forme d'arbre DOM et ensuite ajouter le nouvel élément à l'endroit voulu.

Voici les principales instructions exploitées pour cette manipulation :

- load(filename) obtention d'un objet domDocument à partir d'un fichier
- documentElement() récupération du noeud racine
- getElementsByTagname(nodename) récupération de noeuds sur base de leur nom
- appendChild(new_node) ajout d'un enfant à un noeud
- save(filename) création d'un document XML depuis la représentation DOM

Procédure d'ajout d'un noeud dans un arbre DOM

```
1  - <?php
2  - Class books extends domDocument
3  - {
4  - function __construct()
5  - {
6  - parent::__construct();
7  - }
8  -
9  - function addBook($title, $author, $editor)
10 - {
11 - $eBook = $this->createElement("livre");
12 - $eTitle = $this->createElement("titre");
13 - $eTitle->appendChild($this->createTextNode($title));
14 - $eBook->appendChild($eTitle);
15 - $eAuthor = $this->createElement("auteur");
16 - $eAuthor->appendChild($this->createTextNode($author));
17 - $eBook->appendChild($eAuthor);
18 - $eEditor = $this->createElement("editeur");
19 - $eEditor->appendChild($this->createTextNode($editor));
20 - $eBook->appendChild($eEditor);
21 -
22 - $books=$this->documentElement->getElementsByTagName("livres");
23 - $books->item(0)->appendChild($eBook);
24 - }
25 - }
26 -
27 - header("Content-type: text/xml");
28 - $dom = new books();
29 - $dom->formatOutput = true;
30 - $dom->load("biblio.xml");
31 - $dom->addBook("Lune sanglante", "James Ellroy", "Rivages");
32 - print $dom->saveXML();
33 - $dom->save("biblio.xml");
34 - ?>
```

Manipulation DOM

Opérations DOM - Module DOM PHP5 - Validation d'un document XML

On parle de document XML bien formé quand il respecte bien toutes les règles de syntaxes XML. Mais on parle de document valide, lorsqu'un document est bien formé et qu'en plus il respecte un grammaire donnée : la DTD (Document Type Definition).

La DTD définit les entités et les éléments (ainsi que leurs attributs) destinés à être repris dans un fichier XML. Elle peut être précisée de 2 façons:

1. Sous forme interne, c'est-à-dire en incluant les règles au sein même du document. La référence à une DTD se fait alors dans le prologue du document XML ; on utilise pour cela la balise <!DOCTYPE>.

Document XML avec DTD

```
1  - <?xml version="1.0" encoding="ISO-8859-1"?>
2  - <!DOCTYPE bibliographie [
3  - <!ELEMENT bibliographie (livre*)>
4  - <!ELEMENT livre (titre, auteur, editeur, lieu, date, pages)>
5  - <!ELEMENT titre (#PCDATA)>
6  - <!ELEMENT auteur (#PCDATA)>
7  - <!ELEMENT editeur (#PCDATA)>
8  - <!ELEMENT lieu (#PCDATA)>
9  - <!ELEMENT date (#PCDATA)>
10 - <!ELEMENT pages (#PCDATA)>
11 - ] >
12 -
13 - <bibliographie>
14 - <livre>
15 - <titre>Clandestin</titre>
16 - <auteur>James Ellroy</auteur>
17 - <editeur>Rivages</editeur>
18 - <lieu>Paris</lieu>
19 - <date>1990</date>
20 - <pages>445</pages>
21 - </livre>
22 - <livre>
23 - <titre>Le Dahlia noir</titre>
24 - <auteur>James Ellroy</auteur>
25 - <editeur>Rivages</editeur>
26 - <lieu>Paris</lieu>
27 - <date>1990</date>
28 - <pages>472</pages>
29 - </livre>
30 -
31 - </bibliographie>
```

Définition d'une DTD embarquée dans un fichier de données XML.

Document XML lié à une DTD externe

```
1  - <?xml version="1.0" encoding="ISO-8859-1"?>
2  - <!DOCTYPE bibliographie SYSTEM "biblio.dtd">
3  -
4  - <bibliographie>
5  - <livre>
6  - <titre>Clandestin</titre>
7  - <auteur>James Ellroy</auteur>
```

```
<editeur>Rivages</editeur>
<lieu>Paris</lieu>
<date>1990</date>
<pages>445</pages>
</livre>
<livre>
<titre>Le Dahlia noir</titre>
<auteur>James Ellroy</auteur>
<editeur>Rivages</editeur>
<lieu>Paris</lieu>
<date>1990</date>
<pages>472</pages>
</livre>

</bibliographie>
```

Définition d'une DTD dans un fichier externe.

En PHP, à travers l'API DOM, il est possible de vérifier la validité d'un document XML. On utilise pour cela la méthode 'validate' de l'objet 'DOMDocument'.

PHP et validation DTD

```
<?php
$dom = new DOMDocument;
$dom->Load('biblio.xml');
if ($dom->validate())
   {
   echo "Ce document est valide !\n";
   }
?>
```

Il est également possible de pousser plus avant la phase de validation en ayant recours non plus aux DTD mais aux schémas XML. Ceux-ci, définis par le W3C comme une alternative aux DTD, sont écrits en XML et permettent de préciser un ensemble de définitions et de contraintes appliquables aussi bien sur le contenu que sur la structure même d'un document XML. En fait, les schémas XML enrichissent le concept de la DTD, en permettant de définir des types pour les données.

Schéma correspondant à la bibliographie - biblio.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="bibliographie">
<xs:complexType>
<xs:sequence>
   <xs:element name="livre" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="titre" type="xs:string"/>
      <xs:element name="auteur" type="xs:string"/>
      <xs:element name="editeur" type="xs:string"/>
      <xs:element name="lieu" type="xs:string"/>
      <xs:element name="date" type="xs:gYear"/>
      <xs:element name="pages" type="xs:short"/>
   </xs:sequence>
   </xs:complexType>
   </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
21 - </xs:schema>
```

Mis au point pour pallier aux manquements des DTD, les schémas offrent plus de possibilités que les DTD:
- le typage des données
- l'héritage
- le support des espaces de nom
- la conception par module

Ici encore, l'API DOM de PHP va permettre de vérifier le contenu d'un document XML par rapport à un schéma. On utilise pour cela la méthode 'schemaValidate' de l'objet 'DOMDocument'.

PHP et validation par rapport à un schéma

```
1  - <?php
2  - $dom = new DOMDocument;
3  - $dom->Load('biblio.xml');
4  - if ($dom->schemaValidate('biblio.xsd'))
5  -    {
6  -    echo "Ce document est valide !\n";
7  -    }
8  - ?>
```

Manipulation DOM

Opérations DOM - Module DOM PHP5 - Parcours d'un arbre DOM

L'API DOM est ici employée dans une procédure de parsing qui va extraire des données d'un document XML et les structurer en blocs d'information qui soient aisés à manipuler et à traiter dans le script.

Exemple 1

A partir d'un objet DOMDocument, utilisation de la méthode 'getElementsByTagName' afin de récupérer un ensemble de noeuds 'text'.

La méthode 'getElementsByTagName' retourne une nouvelle instance de la classe DOMNodeList contenant les éléments dont les noms de balises correspondent au paramètre qui a été précisé.

Récupération d'une liste de noeuds

```
1  - <?php
2  - $biblio = new DOMDocument();
3  - $biblio->load('../xml/biblio.xml');
4  -
5  - $titres = $biblio->getElementsByTagName('titre');
6  - foreach ($titres as $titre)
7  - {
8  - $val = $titre->nodeValue;
9  - print utf8_decode($val) . "<br/>";
10 - }
11 - ?>
```

Récupération et affichage des noeuds 'texte' pour les titres contenus dans le document 'biblio.xml'.

Exemple 2

Utilisation de la méthode 'getElementsByTagName' et de 'childNodes' qui retourne tous les fils d'un noeud..

Récupération d'une liste de noeuds

```
1  - <?php
2  - $biblio = new DOMDocument();
3  - $biblio->load('../xml/biblio.xml');
4  -
5  - $livres = $biblio->getElementsByTagName('livre');
6  - foreach ($livres as $livre)
7  - {
8  - foreach ($livre->childNodes as $item)
9  - {
10 - if ($item->nodeName == 'titre')
11 - print utf8_decode($item->nodeValue) . "<br/>";
12 - }
13 - }
14 - ?>
```

Récupération et affichage des noeuds 'texte' pour les titres contenus dans le document 'biblio.xml'

Exemple 3

Utilisation de la méthode 'getElementsByTagName' en combinaison avec 'item'.

La méthode 'item' retourne un noeud spécifié par son index dans un objet DOMNodelist.

Récupération d'une liste de noeuds

```
1  - <?php
2  - $biblio = new DOMDocument();
3  - $biblio->load('../xml/biblio.xml');
4  - 
5  - $livres = $biblio->getElementsByTagName('livre');
6  - foreach ($livres as $livre)
7  - {
8  - $titres = $livre->getElementsByTagName('titre');
9  - if(eregi('prince', $titres->item(0)->nodeValue))
10 - {
11 - $auteur=$livre->getElementsByTagName('auteur');
12 - print $auteur->item(0)->nodeValue . "<br/>";
13 - }
14 - }
15 - ?>
```

Récupération et affichage des noeuds 'texte' pour les auteurs des livres dont le titre contient le mot 'prince'.

Manipulation DOM

Opérations DOM - Module DOM PHP5 - Exploitation d'XPATH

L'exploitation d'XPATH via l'extension DOM est encore plus directe que dans son prédécesseur 'DOM XML'. Pour cela il suffit d'initialiser un objet DOMXpath à partir d'un DOMDocument et d'évaluer l'expression XPATH à l'aide de la méthode 'query'. Cette dernière retourne un objet DOMNodelist contenant tous les noeuds qui correspondent à expression qui a été évaluée.

Utilisation d'XPATH

```
1  - <?php
2  - $dom = new DOMDocument();
3  - $dom->loadHTMLFile("http://www.w3.org/");
4  - $xp = new DOMXpath($dom);
5  - $result = $xp->query("/html/head/title");
6  - print $result->item(0)->firstChild->data;
7  - $xp = new DOMXpath($dom);
8  - $result = $xp->query("/html/body/h2[1]");
9  - print $result->item(0)->firstChild->data;
10 - ?>
```

Récupération du titre d'entête et du premier titre de niveau 2 d'une page au format HTML.

Manipulation SAX

Mode de fonctionnement SAX - Approche événementielle

Avec l'approche SAX, un document XML est lu et analysé par un parseur qui va générer, à chaque fois qu'il rencontre une construction XML, un événement. Cet événement peut être intercepté et traité par le programme via des fonctions ou des méthodes. A partir de là, il est possible de parcourir l'entièreté du document XML et d'en récupérer les composants.

Approche événementielle SAX

Exemple de document XML:

```
<?xml version="1.0"?>
<auteurs>
  <nom>James Ellroy</nom>
  <nom>Jack London</nom>
  <nom>Jules Vernes</nom>
  <nom>Umberto Eco</nom>
  <nom>Dan Simmons</nom>
</auteurs>
```

Analyse SAX :

```
start document
    |
    |-- start element 'auteurs'
    |         |
    |         |-->start element 'nom'
    |         |         |
    |         |         |-->characters 'James Ellroy'
    |         |
    |         |-->end element 'nom'
    |         |
    |         |-->start element 'nom'
    |         |         |
    |         |         |-->characters 'Jack London'
    |         |
    |         |-->end element 'nom'
    |         |
    |         |-->start element 'nom'
    |         |         |
    |         |         |-->characters 'Jules Vernes'
    |         |
    |         |-->end element 'nom'
    |         |
    |         |-->start element 'nom'
    |         |         |
```

```
    |         |         |-->characters 'Umberto Eco'
    |         |
    |         |-->end element 'nom'
    |         |
    |         |-->start element 'nom'
    |         |         |
    |         |         |-->characters 'Dan Simmons'
    |         |
    |         |-->end element 'nom'
    |
    |-- end element 'auteurs'
    |
end document
```

L'analyse SAX repose sur la prise en compte d'événements : début du document, fin du document, début d'un élément, fin d'un élément ... C'est au programmeur à tirer profit de l'apparition ou non de ces événements afin de retirer du document l'information qui l'intéresse.

PHP est muni d'un parseur de type SAX. Celui-ci est basé sur la librairie 'expat' créée par James Clark. Voyons comment tirer parti de cet outil.

Manipulation SAX

Mode de fonctionnement SAX - Initialisation du parseur

L'initialisation du parseur se fait via la fonction prédéfinie 'xml_parser_create()'. Celle-ci retourne un 'handle' pour le parseur SAX.

Syntaxe : `xml_parser_create()`

initialisation du parseur SAX

```
1  - <?php
2  - $parser = xml_parser_create();
3  - ?>
```

La fonction 'xml_parser_create()' retourne un 'handle' pour le parseur SAX. Ce handle est représenté par une variable '$parser'.

Manipulation SAX

Mode de fonctionnement SAX - Définition des gestionnaires d'événements

Une fois le handle de parseur obtenu, on peut définir les événements significatifs que l'on désire intercepter. Pour cela, on utilisera notamment les instructions suivantes:

- xml_set_element_handler()
- xml_set_character_data_handler()

xml_set_element_handler() accepte trois arguments : le handle de parseur, le nom de la fonction à appeler lorsqu'une balise d'ouverture est trouvée, et le nom de la fonction à appeler lorsqu'une balise de fermeture est trouvée.

Syntaxe : `xml_set_element_handler($xml_parser, "startElement" "endElement")`

xml_set_character_data_handler() accepte deux arguments : le handle de parseur, et le nom de la fonction à appeler lorsque le parseur trouve des contenus textuels.

Syntaxe : `xml_set_character_data_handler($xml_parser, "characters")`

Gestionnaires d'événements SAX

```
1  - <?php
2  - $parser = xml_parser_create();
3  - xml_set_element_handler($parser, "startElement", "endElement");
4  - xml_set_character_data_handler($parser, "characters");
5  - ?>
```

Les instructions xml_set_element_handler et xml_set_character_data_handler instancient les gestionnaires d'événements: les débuts et fins de balises ainsi que les noeuds textes seront traités par les fonctions 'callback' ('startElement', 'endElement', 'characters').

Manipulation SAX

Mode de fonctionnement SAX - Définition des fonctions 'callback'

Les gestionnaires d'événements 'xml_set_element_handler' et 'xml_set_character_data_handler' faisant référence à des fonctions, il nous faut à présent définir celles-ci.

Fonctions liées à 'xml_set_element_handler'

Lorsqu'une balise d'ouverture est rencontrée par le parseur, il fait appel à la fonction enregistrée dans le gestionnaire 'xml_set_element_handler' (en ce qui nous concerne elle s'appelle 'startElement'). Cette fonction reçoit automatiquement trois paramètres : le handle de parseur xml, le nom de l'élément courant, la liste des attributs de l'élément (sous la forme d'un tableau associatif).

Parallèlement, lorsqu'une balise de fermeture est rencontrée par le parseur, il fait appel à la fonction enregistrée dans le gestionnaire 'xml_set_element_handler' (en ce qui nous concerne elle s'appelle 'endElement'). Cette fonction reçoit automatiquement deux paramètres : le handle de parseur xml et le nom de l'élément courant.

Fonctions 'startElement' et 'endElement'

```
1  - <?php
2  - function startElement($parser, $name, $attributes)
3  - {
4  - echo "<p><b>
5  - $name
6  - </b>";
7  - }
8  -
9  - function endElement($parser, $name)
10 - {
11 - echo "</p>";
12 - }
13 -
14 - $parser = xml_parser_create();
15 - xml_set_element_handler($parser, "startElement", "endElement");
16 - ...
17 - ?>
```

La fonction 'startElement' se contente ici d'afficher le nom de l'élément courant ($name). Ce nom lui est automatiquement fourni par le gestionnaire d'événements.
La fonction 'endElement' ne reçoit auncun paramètres. Dans cet exemple, elle ne fait que fermer proprement un paragraphe HTML.

Fonction liée à 'xml_set_character_data_handler'

Lorsque le parseur trouve un contenu textuel, il fait appel à la fonction enregistrée dans le gestionnaire 'xml_set_character_data_handler' (en ce qui nous concerne elle s'appelle characters'). Cette fonction reçoit automatiquement deux paramètres : le handle de parseur xml et la chaîne de caractères récupérée.

Fonction 'characters'

```
1  - <?php
2  - ...
3  - function characters($parser, $data)
4  - {
5  - echo "<blockquote>$data</blockquote>";
6  - }
7  -
8  - $parser = xml_parser_create();
9  - xml_set_element_handler($parser, "startElement" "endElement");
10 - xml_set_character_data_handler($parser, "characters");
11 - ...
12 - ?>
```

Dans cet exemple, la fonction 'characters' affiche le contenu des noeuds de type 'texte' qui ont été relevés via le gestionnaire d'événements.

Manipulation SAX

Mode de fonctionnement SAX - Lancement du parseur

Pour lancer la procédure d'analyse proprement dite, on utilise la fonction prédéfine 'xml_parse'. C'est elle qui lance et active la parseur SAX.

Syntaxe : `xml_parse(parser_handle, xml_data)`

Cette fonction prend deux paramètres : le handle de parseur précédemment obtenu via l'instruction 'xml_parser_create', les données XML. En réponse, 'xml_parse' retourne une valeur boléenne : 'true' si l'analyse se déroule correctement et 'false' dans le cas contraire.

Si les données XML se trouvent dans un fichier, il faut les lire (par exemple avec la fonction 'fread') avant de les envoyer à la fonction 'xml_parse'. La lecture se fait par "morceau", ainsi il n'est pas nécessaire de charger tout le document en mémoire.

Lancement de l'analyse XML

```
1  - <?php
2  - # définition des fonctions callback
3  - ...
4  -
5  - # initialisation du parseur
6  - $parser = xml_parser_create();
7  -
8  - # gestionnaires d'événements
9  - ...
10 -
11 - # ouverture en lecture du fichier
12 - $xml_file = "../xml/biblio.xml";
13 - if (!($fp = fopen($xml_file, "r")))
14 - {
15 - die("Impossible d'ouvrir le fichier : $xml_file");
16 - }
17 -
18 - # lecture et analyse des données
19 - while ($xml_data = fread($fp, 4096))
20 - {
21 - if (!xml_parse($xml_parser, $xml_data))
22 -    {
23 -    die("Une erreur a été détectée dans le fichier XML!");
24 -    }
25 - }
26 -
27 - # fermeture du handle de parseur
28 - xml_parser_free($parser);
29 - ?>
```

Si le document XML n'est pas bien formé, 'xml_parse' retourne 'false' et le traitement est interrompu. Si par contre tout se passe correctement, les gestionnaires d'événements ainsi que les fonctions 'callback' sont pris en compte et appelés autant de fois que nécessaire.

Dans les articles suivants, nous verrons comment -concrètement- mettre en oeuvre les instructions de l'interface SAX.

Manipulation SAX

Opérations SAX - Vérification d'un document XML

L'utilisation d'un parseur SAX va permettre de vérifier si un document XML est bien formé. Si ce n'est pas le cas, le parseur SAX va renvoyer un message d'erreur qu'il est possible de récupérer et de décoder.

Voici les instructions PHP liées à l'interface SAX que nous allons utiliser pour effectuer cette vérification :

- xml_parser_create() initialisation d'un parseur et obtention d'un handle de parseur
- xml_parse(sax_handler, xml_data) lancement de l'analyse SAX
- xml_get_error_code(sax_handler) récupération du code d'erreur
- xml_error_string(error_code) récupération de la description de l'erreur
- xml_get_current_line_number(sax_handler) récupération du numéro de la ligne courante

Vérification d'un document XML

```
1  - <?php
2  - # initialisation du parseur
3  - $parser = xml_parser_create();
4  -
5  - # ouverture en lecture du fichier
6  - $xml_file = "../xml/biblio.xml";
7  - if (!($fp = fopen($xml_file, "r")))
8  - {
9  - die("Impossible d'ouvrir le fichier : $xml_file");
10 - }
11 -
12 - # lecture et analyse des données
13 - while ($xml_data = fread($fp, 4096))
14 - {
15 -    if (!xml_parse($parser, $xml_data))
16 -    {
17 -    $error = xml_get_error_code(parser);
18 -    die("Une erreur a été détectée dans le fichier XML:<br>" .
19 -    "- code d'erreur : $error<br>" .
20 -    "- explication : \"" . xml_error_string($error) . "\" " .
21 -    "à la ligne " . xml_get_current_line_number(parser));
22 -    }
23 - }
24 -
25 - xml_parser_free($xml_parser);
26 - ?>
```

La fonction 'xml_parser_create()' retourne un 'handle' pour le parseur SAX. Ce handle est représenté par la variable '$parser'.
Si le document XML n'est pas bien formé, 'xml_parse' retourne 'false' et le bloc d'instructions de traitement de l'erreur est exécuté.
Le code d'erreur est récupéré grâce à 'xml_get_error_code'. Ce code est passé à la fonction 'xml_error_string' afin d'obtenir une description plus explicite. Enfin, la ligne à laquelle a été constatée l'erreur est fournie par 'xml_get_current_line_number'.

Manipulation SAX

Opérations SAX - Récupération d'une structure XML

Le mode de fonctionnement de SAX permet de parcourir très rapidement toute la structure d'un document XML. Le profile du parcours est déterminé par les gestionnaires d'évenements et les fonctions 'callback'.

Dans ce chapitre, nous allons utiliser SAX afin de récupérer (et présenter à l'écran) tous les noeuds éléments et les noeuds textes d'un document XML.

Voici les instructions PHP liées à l'interface SAX que nous allons utiliser pour récupérer ces éléments :

- xml_set_element_handler(SAX_handler, start_func, end_func) gestionnaire concernant les noeuds éléments
- xml_set_character_data_handler(SAX_handler, char_func) gestionnaire concernant les noeuds textes
- xml_parser_create() initialisation d'un parseur et obtention d'un handle de parseur
- xml_parse(sax_handler, xml_data) lancement de l'analyse SAX
- xml_parser_free(sax_handler) libération du handle de parseur

Parcours d'une structure XML

```
1  - <?php
2  - # fonction liée aux débuts de balises
3  - function startElement($parser, $name, $attributes)
4  - {
5  - echo "<p><b>$name</b>";
6  - }
7  -
8  - # fonction liée aux fins de balises
9  - function endElement($parser, $name)
10 - {
11 - echo "</p>";
12 - }
13 -
14 - # fonction liée aux noeuds textes
15 - function characters($parser, $data)
16 - {
17 - echo "<blockquote>$data</blockquote>";
18 - }
19 -
20 - # initialisation du parseur
21 - # obtention d'un handle de parseur
22 - $parser = xml_parser_create();
23 -
24 - # gestionnaires d'événements
25 - xml_set_element_handler($parser, "startElement", "endElement");
26 - xml_set_character_data_handler($parser, "characters");
27 -
28 - # ouverture en lecture du fichier
29 - $xml_file = "../xml/biblio.xml";
30 - if (!($fp = fopen($xml_file, "r")))
31 - {
32 - die("Impossible d'ouvrir le fichier : $xml_file");
33 - }
34 -
35 - # lecture et analyse des données
36 - while ($xml_data = fread($fp, 4096))
37 - {
38 -    if (!xml_parse($parser, $xml_data))
39 -    {
40 -    # message d'erreur
41 -    die("Une erreur a été détectée dans votre fichier XML!");
42 -    }
43 - }
44 -
45 - # fermeture du handle de parseur
```

```
46 - xml_parser_free($parser);
47 - ?>
```

La fonction 'xml_parser_create()' initialise le parseur SAX et crée un handle de parseur (celui-ci pourra être utilisé dans d'autres fonctions par la suite).
Une fois le handle de parseur obtenu, on définit les gestionnaires d'événements que l'on veut exploiter. Ces gestionnaires font référence aux fonctions 'callback' définie au début du script.
La fonction 'xml_parse' lance l'analyse. Elle appelera les fonctions 'callback' appropriées en fonction des éléments XML qu'elle rencontrera.

A chaque fois que le parseur rencontre une balise d'ouverture, il appelle la fonction 'startElement' en lui passant le nom de la balise (et les éventuels attributs). La fonction 'startElement' traite la balise en exécutant les instructions qu'elle contient.

La fonction 'endElement' s'occupe, elle, des balises de fermeture et se comporte de façon similaire à 'startElement' (excepté qu'elle ne reçoit pas d'attributs).

Les noeuds 'texte' seront eux traités par la fonction 'characters'.

C'est au programmeur à aménager ces différentes fonctions afin qu'elles servent au mieux ses objectifs. Ainsi, l'approche SAX permet de gérer uniquement les éléments pertinents sans avoir à construire en mémoire une structure contenant l'intégralité du document.

Traitement d'un flux RSS

```
1  - <?php
2  - $listItems = array();
3  - $countItems = 0;
4  - $curTag = '';
5  - $title = '';
6  - $descr = '';
7  -
8  - function startElement($xp,$name,$attributes)
9  - {
10 - global $countItems, $curTag, $title, $description;
11 - $curTag = "$name";
12 - if ($name == "ITEM")
13 -    {
14 -    $countItems++;
15 -    $title = "";
16 -    $description = "";
17 -    }
18 - }
19 -
20 - function endElement($xp,$name)
21 - {
22 - global $title, $description, $listItems;
23 - if ($name == "ITEM")
24 -    {
25 -    $record = array("title" => $title, "descr" => $descr);
26 -    $listItems[] = $record;
27 -    }
28 - }
29 -
30 - function characterDataHandler($xp,$data)
31 - {
32 - global $curTag, $title, $descr;
33 - switch($curTag)
34 -    {
35 -    case "TITLE":
36 -       $title .= $data;
37 -       break;
38 -    case "DESCRIPTION":
39 -       $descr .= $data;
```

```
40 -       break;
41 -    }
42 - }
43 -
44 -
45 - $fh = fopen('sampledata/xml/test1.rss','r');
46 - # Ouverture du fichier XML
47 - $xp = xml_parser_create();
48 - # Initialisation du parseur XML
49 - xml_set_element_handler($xp, "startElement", "endElement");
50 - xml_set_character_data_handler($xp, "characterDataHandler");
51 - # Définition des fonctions à appeler
52 -
53 - while ($data = fread($fh, 4096))
54 - {
55 - if (!xml_parse($xp,$data)) return 'Error in the feed';
56 - }
57 -
58 - foreach ($listItems as $item)
59 - {
60 - echo "Article :<br/>\n";
61 - foreach ($item as $key => $value)
62 -    echo "$key : $value <br/>\n";
63 - echo "--------------------------<br/><br/>\n\n";
64 - }
65 - ?>
```

Dans cet exemple, le déclenchement des événements SAX au cours de l'analyse de notre document au format RSS va se faire pour les balises 'ITEM' et (en second niveau) pour les balises 'TITLE' et 'DESCRIPTION'.

Manipulation SimpleXML

Charger une structure XML

SimpleXML est un module standard de PHP5 qui permet de manipuler et de parcourir aisément des structures XML relativement simples.

Pour charger un document XML avec ce module, il suffit de lancer la fonction 'simplexml_load_file()'. Celle-ci prend comme argument le chemin d'accès du fichier XML et retourne un objet de la classe 'simplexml_element'.

Syntaxe : `simplexml_load_file(file_path)`

Chargement d'un document XML

```
1  - <?php
2  - $root = simplexml_load_file("./biblio.xml");
3  - ?>
```

L'objet 'simplexml_element' qui est retourné représente l'élément racine du document XML pris en compte.

Pour charger, non pas un document XML, mais une chaîne de caractères au format XML il faut utiliser la fonction 'simplexml_load_string()'. Celle-ci prend comme argument un flux XML sous forme d'une chaîne de caractères et retourne comme précédement un objet de la classe 'simplexml_element'.

Syntaxe : `simplexml_load_string(string_xml)`

Chargement d'un flux XML

```
1  - <?php
2  - $xml = "<livre><livre><id>45</id></livre></livres>";
3  - $root = simplexml_load_string($xml);
4  - ?>
```

Manipulation SimpleXML

Manipuler les éléments XML

Après avoir charger un document XML, il est possible d'en traiter les éléments simplement en interrogeant les propriétés de l'objet 'simplexml_element' obtenu via la fonction 'simplexml_load_file()'.

Récupération d'un noeud

Lorsqu'il s'agit d'un document XML simple, pour accéder à un noeud à partir de la racine il s'uffit de l'appeler par son nom. Prenons pour exemple un document XHTML dont on veur récupérer le tritre de niveu 1 :

Récupération d'un noeud avec SimpleXML

```
1  - <?php
2  - if (file_exists('test.html'))
3  - {
4  - $root = simplexml_load_file("test.html");
5  - $body = $root->body;
6  - echo $body->h1;
7  - }
8  - ?>
```

Récupération de plusieurs noeuds

Toujours en les appelant par leur nom, mais cette fois-ci en parcourant un tableau (array), on va pouvoir afficher le contenu des différents paragraphes de notre document XML.

Récupération de plusieurs noeuds avec SimpleXML

```
1  - <?php
2  - if (file_exists('test.html'))
3  - {
4  - $root = simplexml_load_file("test.html");
5  - $body = $root->body;
6  - foreach($body->p as $p)
7  - {
8  - echo utf8_decode($p) . "<br/><br/>";
9  - }
10 - }
11 - ?>
```

Lecture d'un flux RSS avec SimpleXML

```
1  - <?php
2  - $root = simplexml_load_file("test.rss");
3  - foreach($root->channel->item as $item)
4  - {
```

```
5  - echo "News : " . utf8_decode($item->title) . "<br/>";
6  - }
7  - }
8  - ?>
```

Récupération d'un attribut

Pour récupérer un attribut, on utilise la technique d'interrogation des tableaux associatifs :

Récupération d'un attribut

```
1  - <?php
2  - if (file_exists('biblio.xml'))
3  - {
4  - $root = simplexml_load_file("biblio.xml");
5  - $books = $root->books;
6  - foreach($books->item as $item)
7  - {
8  - echo $item['id'] . "<br/>";
9  - }
10 - }
11 - ?>
```

La valeur renvoyée par cet appale sera le contenu de l'attribut 'id' des éléments 'item'.

Récupération de plusieurs attributs

La méthode 'attributes()' du module SimpleXML renvoie un tableau associatif contenant la clé et la valeur de chaque attribut d'un élément. Il suffit ensuite des les parcourir à l'aide d'une boucle :

Récupération de plusieurs attributs

```
1  - <?php
2  - $root = simplexml_load_file("biblio.xml");
3  - $books = $root->books;
4  - foreach($books->item as $item)
5  - {
6  - echo "Element : " . $item['id'] . "<br/>";
7  - echo "Attributes : <br/>";
8  - foreach($item->attributes as $key => $val)
9  - {
10 - echo $key . "=" . $val . "<br/>";
11 - }
12 - }
13 - ?>
```

Affichage pour chaque élément 'item' du nom et de la valeur de tous ses attributs.

Echange WDDX

Sérialisation au format WDDX

WDDX (Web Distributed Data Exchange) est une implémentation du langage XML permettant de décrire de manière générique des structures de données, telles que les tableaux scalaires et associatifs, pour pouvoir les transférer d'une application à une autre. Différentes platesformes mettent en oeuvre WDDX : ColdFusion, Active Server Pages, JavaScript, Perl, Java, Python, Macromedia Flash et PHP.

Le principe de fonctionnement suivi est celui de la sérialisation. Il consiste à pouvoir prendre une structure en mémoire et à en sauvegarder l'état sur un flux de données (fichier ou chaîne de caractères, par exemple). Ce méanisme permettra également de reconstruire, dans un deuxième temps, la structure en mémoire à l'identique de ce qu'elle pouvait être à l'origine. Ainsi, les outils de sérialisation sont utilisés pour transmettre ou stocker des structures de données plus ou moins complexes.

Depuis sa version 3, PHP offre la possibilité d'exploiter le format WDDX. Voici les principales fonctions liées à cette extension :

Syntaxe : `wddx_serialize_vars(var_name [, var_name])`

Cette fonction sérialise la valeur de une ou plusieurs variables dans un paquet WDDX. Elle prend en argument le nom des structures à enregistrer dans le paquet. En réponse, elle retourne une chaîne de caractères au format WDDX.

Syntaxe : `wddx_deserialize(packet)`

Une fois le paquet WDDX obtenu, il est possible de le désérialiser an lui appliquant la fonction 'wddx_deserialize'. Celle-ci retourne la stucture de données initiale.

Sérialisation WDDX

```
1  - <?php
2  - $villes = array('Bruxelles', 'Namur', 'Anvers', 'Gand');
3  - $tmp = wddx_serialize_vars('villes');
4  -
5  - echo $tmp;
6  - ?>
```

Sérialisation XML d'un tableau et affichage du résultat (voir plus bas).

Résultat d'une sérialisation WDDX

```
1  - <wddxPacket version='1.0'>
2  - <header/>
3  - <data>
4  - <struct>
5  - <var name='villes'>
6  - <array length='4'>
```

```
7  - <string>Bruxelles</string>
8  - <string>Namur</string>
9  - <string>Anvers</string>
10 - <string>Gand</string>
11 - </array>
12 - </var>
13 - </struct>
14 - </data>
15 - </wddxPacket>
```

Désérialisation WDDX

```
1  - <?php
2  - $villes = array('Bruxelles', 'Namur', 'Anvers', 'Gand');
3  - # Sérialisation
4  - $tmp = wddx_serialize_vars('villes');
5  -
6  - # Désérialisation
7  - $data = wddx_deserialize($tmp);
8  - # Parcours du tableau récupéré
9  - foreach($data['villes'] as $ville)
10 -    {
11 -    echo $ville . "";
12 -    }
13 - ?>
```

L'échange WDDX est très pratique pour transmettre des tableaux : son point fort réside dans le fait qu'il est compatible avec de nombreux langages, alors que la sérialisation classique (avec les fonctions 'serialize' et 'unserialize') se limite à de la communication entre scripts PHP uniquement.

Exemples et mise en pratique

Transformation XSLT

Transformation avec le parseur externe XT

L'exercice consiste à utiliser un parseur externe (en l'occurence 'xt') pour réaliser une transformation XSLT dynamique à partir d'un navigateur Web.

Script appelé par le navigateur ('biblio.php')

```
1  - <?php
2  - include("tools.php");
3  - xslt_transform("../xml/biblio.xml", "../xsl/biblio-xt.xsl");
4  - ?>
```

Ce script commence par inclure un fichier php servant de boîte à outils 'tools.php'. Il appelle ensuite la fonction 'xslt_transform()' dont la définition se trouve dans 'tools.php'.
Deux paramètres sont passés à la fonction 'xslt_transform()' : le chemin d'accès relatif du fichier XML et celui du fichier XSL.

Script inclus 'tools.php'

```
1  - <?php
2  - function xslt_transform($fichierxml, $fichierxsl)
3  - {
4  - # Parameters #
5  - $params = "";
6  - while (list($key, $val) = each($_GET))
7  -    {
8  -    $params .= "$key=$val ";
9  -    }
10 -
11 - # Parser XSLT (external tool) #
12 - passthru("xt.exe $fichierxml $fichierxsl $params");
13 - }
14 - ?>
```

Ce script contient la définition de la fonction 'xslt_transform()'. Celle-ci récupère les paramètres qui lui sont envoyés par la méthode 'GET' et les prépare pour les passer au parseur externe 'xt'.
Le programme 'xt' s'attend à ce que les paramètres éventuels lui soient transmis sous forme de couple "nom_du_paramètre=valeur_du_paramètre".
La dernière instruction de la définition de fonction active le parseur 'xt'. Ceci se fait via la fonction prédéfinie 'passthru()'. Elle prend un seul paramètre : une chaîne de caractère qui doit correspondre à un appel classique au programme 'xt'.

Transformation XSLT

Transformation avec l'extension Sablotron

L'exercice consiste à réaliser une transformation XSLT dynamique via l'extension xslt de PHP (Sablotron).

Script appelé par le navigateur ('biblio.php')

```
1  - <?php
2  - include("tools.php");
3  - xslt_transform("htdocs/xml/biblio.xml",
   "htdocs/xsl/biblio-sablot.xsl");
4  - ?>
```

Ce script commence par inclure un fichier php servant de boîte à outils 'tools.php'. Il appelle ensuite la fonction 'xslt_transform()' dont la définition se trouve dans 'tools.php'.
Deux paramètres sont passés à la fonction 'xslt_transform()' : le chemin d'accès du fichier XML et celui du fichier XSL. Ces chemins d'accès sont construits à partir du répertoire de référence de l'extension xslt de PHP, à savoir le répertoire d'installation d'Apache. Si l'on veut ne pas dépendre de ce répertoire on précisera les chemins d'accès en faisant explicitement référence au protocole "file" (exemple : file:///documents/xmlfiles/biblio.xml).

Script inclus 'tools.php'

```
1  - <?php
2  - function xslt_transform($fichierxml, $fichierxsl)
3  - {
4  - # Parameters #
5  - $params = array();
6  - $args = array();
7  -
8  - while (list($key, $val) = each($_GET))
9  -    {
10 -    $params[$key] = utf8_encode($val);
11 -    }
12 -
13 - # Parser XSLT (Sablotron) #
14 - $prc = xslt_create();
15 - $result = xslt_process($prc, "$f_xml", "$f_xsl", NULL, $args,
   $params);
16 - echo $result;
17 - ?>
```

Ce script contient la définition de la fonction 'xslt_transform()'. Celle-ci récupère les paramètres qui lui sont envoyés par la méthode 'GET' et les prépare pour les passer à l'instruction 'xslt_process()'.
L'instruction 'xslt_process()' s'attend à ce que les paramètres éventuels lui soient transmis sous forme de tableau associatif. L'utilisation de la fonction 'utf8_encode' (conversion d'une chaîne ISO-8859-1 en UTF-8) dans la procédure de récupération des paramètres assure une bonne compréhension des caractères accentués par le parseur XSLT.
La fonction 'xslt_create()' initialise un processeur XSLT et 'xslt_process()' lance la transformation. 'xslt_process()' prend comme paramètres le handle du processeur XSLT ($processor), le chemin d'accès du fichier XML ($f_xml), le chemin d'accès du fichier XSL ($f_xsl) et, en dernier lieu, les paramètres ($params).

Transformation XSLT

Transformation avec l'extension DOM XML

L'exercice consiste à réaliser une transformation XSLT dynamique via l'implémentation DOM de PHP.

Script appelé par le navigateur ('biblio.php')

```
1  - <?php
2  - include("tools.php");
3  - xslt_transform(realpath("biblio.xml"),
   relapath("biblio-domxml.xsl"));
4  - ?>
```

Ce script commence par inclure un fichier php servant de boîte à outils 'tools.php'. Il appelle ensuite la fonction 'xslt_transform()' dont la définition se trouve dans 'tools.php'.
Deux paramètres sont passés à la fonction 'xslt_transform()' : le chemin d'accès du fichier XML et celui du fichier XSL (avec DOM XML on utilise des chemins d'accès absolus).

Script inclus 'tools.php'

```
1  - <?php
2  - function xslt_transform($fichierxml, $fichierxsl)
3  - {
4  - # Parameters #
5  - $params = array();
6  -
7  - while (list($key, $val) = each($_GET))
8  -    {
9  -    $params[$key] = utf8_encode($val);
10 -    }
11 -
12 - # Parser XSLT (DOM XML) #
13 - $xslt = domxml_xslt_stylesheet_file($fichierxsl);
14 - $xml = domxml_open_file($fichierxml);
15 - $result = $xslt->process($xml, $params);
16 - print $xslt->result_dump_mem($result);
17 - ?>
```

Ce script contient la définition de la fonction 'xslt_transform()'. Celle-ci récupère les paramètres qui lui sont envoyés par la méthode 'GET' et les prépare pour les passer à la méthode 'process()' de l'objet DomXsltStylesheet .
La méthode 'process()' s'attend à ce que les paramètres éventuels lui soient transmis sous forme de tableau associatif. L'utilisation de la fonction 'utf8_encode' (conversion d'une chaîne ISO-8859-1 en UTF-8) dans la procédure de récupération des paramètres assure une bonne compréhension des caractères accentués par le parseur XSLT.
La fonction 'domxml_xslt_stylesheet_file()' crée un objet DomXsltStylesheet qui est représenté par la variable $xslt.
La fonction 'domxml_open_file()' crée un objet DomDocument à partir du fichier XML "biblio.xml". Cet objet est représenté par la variable $xml.
le résultat de la transformation est envoyé sur la sortie standard par l'intermédiare de la méthode "result_dump_mem()". Cette dernière retournant le résultat XSLT sous la forme d'une chaîne de caractères.

Manipulation DOM

Construction d'un document XML

L'exercice consiste à créer un document XML à prtir du modèle DOM et de ses outils. Nous allons ici réaliser un fichier XML qui décrit une bibliographie

Voici le document XML que nous aimerions obtenir via les instructions DOM :

Résultat attendu

```
1  - <?xml version="1.0" ?>
2  - <!-- Test DOM/PHP -->
3  - <bibliographie auteur="Circum Net">
4  - <livres>
5  - <livre>
6  - <titre>Clandestin</titre>
7  - <auteur>James Ellroy</auteur>
8  - <editeur>Rivages</editeur>
9  - </livre>
10 - <livre>
11 - <titre>Le Petit Prince</titre>
12 - <auteur>Antoine de Saint-Exup?</auteur>
13 - <editeur>Gallimard</editeur>
14 - </livre>
15 - <livre>
16 - <titre>Barberousse</titre>
17 - <auteur>Michel Tournier</auteur>
18 - <editeur>Gallimard</editeur>
19 - </livre>
20 - </livres>
21 - </bibliographie>
```

Pour arriver à ce résultat, voici la procédure qu'il faut mettre en oeuvre :

Script appelé par le navigateur ('biblio.php')

```
1  - <?php
2  - $doc = domxml_new_doc("1.0");
3  - $comment = $doc->create_comment("Test DOM/PHP");
4  - $doc->append_child($comment);
5  -
6  - $bibliographie = $doc->create_element("bibliographie");
7  - $bibliographie->set_attribute("auteur", "Circum Net");
8  - $doc->append_child($bibliographie);
9  -
10 - $livres = $doc->create_element("livres");
11 - $bibliographie->append_child($livres);
12 -
13 - $livre = $doc->create_element("livre");
14 - $titre = $doc->create_element("titre");
15 - $titre->append_child($doc->create_text_node("Clandestin"));
16 - $livre->append_child($titre);
17 - $auteur = $doc->create_element("auteur");
18 - $auteur->append_child($doc->create_text_node("James Ellroy"));
19 - $livre->append_child($auteur);
20 - $editeur = $doc->create_element("editeur");
21 - $editeur->append_child($doc->create_text_node("Rivages"));
22 - $livre->append_child($editeur);
23 - $livres->append_child($livre);
24 -
25 - $livre = $doc->create_element("livre");
26 - $titre = $doc->create_element("titre");
```

```
27 - $titre->append_child($doc->create_text_node("Le Petit Prince"));
28 - $livre->append_child($titre);
29 - $auteur = $doc->create_element("auteur");
30 - $auteur->append_child($doc->create_text_node("Inconnu"));
31 - $livre->append_child($auteur);
32 - $editeur = $doc->create_element("editeur");
33 - $editeur->append_child($doc->create_text_node("Gallimard"));
34 - $livre->append_child($editeur);
35 - $livres->append_child($livre);
36 -
37 - $livre = $doc->create_element("livre");
38 - $titre = $doc->create_element("titre");
39 - $titre->append_child($doc->create_text_node("Barberousse"));
40 - $livre->append_child($titre);
41 - $auteur = $doc->create_element("auteur");
42 - $auteur->append_child($doc->create_text_node("Michel Tournier"));
43 - $livre->append_child($auteur);
44 - $editeur = $doc->create_element("editeur");
45 - $editeur->append_child($doc->create_text_node("Gallimard"));
46 - $livre->append_child($editeur);
47 - $livres->append_child($livre);
48 -
49 - echo $doc->dump_mem(true);
50 - ?>
```

Ce script commence par inclure un fichier php servant de boîte à outils 'tools.php'. Il appelle ensuite la fonction 'xslt_transform()' dont la définition se trouve dans 'tools.php'.
Deux paramètres sont passés à la fonction 'xslt_transform()' : le chemin d'accès relatif du fichier XML et celui du fichier XSL.

A partir de là les évolutions possibles sont multiples et l'exercice peut être facilement prolongé :

- construction du document XML à partir d'une base de données;
- adaptation à l'extension DOM de PHP5;
- ...

Manipulation DOM

Outil de recherche DOM

L'exercice consiste à réaliser un outil de recherche sur base des instructions DOM : un mot-clé est saisi dans un formulaire HTML et est recherché dans un fichier XML via les fonctions prédéfinies DOM.

Le script que nous allons réaliser se divise en deux parties. La première partie s'occupe d'afficher un formulaire HTML afin de récupérer un mot-clé. La seconde partie, sur base du mot-clé récupéré, fouille le document XML.

Script appelé par le navigateur ('search.php') - première partie

```
1  - <?php
2  - # fichiers inclus
3  - require('html_tools.php');
4  - require('tools.php');
5  -
6  - # fonction de mise en page (entête)
7  - print_header_page();
8  -
9  - # teste si un mot-clé a déjà été saisi
10 - # sinon : affichage d'un formulaire
11 - if (empty($_GET[keyword]))
12 - {
13 - echo "
14 - <form method=\"get\" action=\"\">
15 - <span >Mot-clé : </span>
16 - <input type=\"text\" size=\"10\" name=\"keyword\">
17 - <input type=\"submit\" value=\"ok\">
18 - </form>
19 - ";
20 - }
21 - ?>
```

Ce script commence par inclure deux fichiers php servant de boîtes à outils 'html_tools.php' et 'tools.php'. Il appelle la fonction 'print_header_page()' dont la définition se trouve dans 'html_tools.php'. Cette fonction génère le code HTML d'entête de page.
Ensuite un test est effetué : si la variable globale '$_GET[keyword]' n'a pas été affectée alors le script affiche un formulaire d'encodage HTML. Ce formulaire est des plus simple puisqu'il ne présente qu'un seul champ de saisie.

Script appelé par le navigateur ('search.php') - seconde partie

```
1  - <?php
2  - {
3  - // parsing du document XML
4  - $xml_file = "D:\xml\biblio.xml";
5  - if(!$doc = domxml_open_file($xml_file))
6  - {
7  - die("Le document XML contient des erreurs !");
8  - }
9  -
10 - // récupération de la racine et des noeuds enfants
11 - $root = $doc->document_element();
12 - $children = $root->get_elements_by_tagname("livre");
13 -
14 - // appel de la fonction qui réalise la recherche
15 - search_and_print($children, $_GET[keyword]);
```

```
16 - }
17 -
18 - print_footer_page();
19 - ?>
```

Cette partie du script est activée si l'utilisateur est passé par le formulaire HTML.
Le script crée un objet DomDocument sur base du document XML fourni. Il récupère tous les noeuds 'livre' à partir de la racine de l'arbre DOM. Ensuite, la fonction 'search_and_print' est appelée. Celle-ci prend comme paramètres la liste des noeuds 'livre' ainsi que le critère de recherche.
La fonction 'search_and_print' se trouve dans le fichier 'tools.php'.

Le script principal 'search.php' fait appel à la fonction 'search_and_print' dont la définition est encodée dans le script inclus 'tools.php'. Cette fonction est destinée à parcourir le document XML et à repérer si les éléments 'titre' contiennent le mot-clé saisi par l'utilisateur.

Script inclus 'tools.php'

```
1  - <?php
2  - function search_and_print($nodelist, $search)
3  - {
4  - for ($i=0; $i<sizeof($nodelist); $i++)
5  - {
6  - $titles = $nodelist[$i]->get_elements_by_tagname("titre");
7  - if (eregi($search,$titles[0]->get_content()))
8  -    {
9  -    $title_str = $titles[0]->get_content();
10 -    $a = $nodelist[$i]->get_elements_by_tagname("auteur");
11 -    $a_str = $a[0]->get_content();
12 -    echo "<p>";
13 -    echo utf8_decode($title_str);
14 -    echo " - ";
15 -    echo utf8_decode($a_str);
16 -    echo "</p>";
17 -    }
18 - }
19 - }
20 - ?>
```

Pour chaque noeud présent dans le tableau ($nodelist) reçu en paramètre, le script récupère l'élément "titre" et teste s'il contient le critère de recherche ($search). Dans l'affirmative, le titre ainsi que l'auteur sont affichés à l'écran.

Manipulation DOM

Outil de recherche XPATH

De la même façon que nous avons réalisé un outil de recherche sur base des instructions DOM, nous allons maintenant nous appuyer sur la norme XPATH pour atteindre un objectif identique : un mot-clé est saisi dans un formulaire HTML et est recherché dans un fichier XML via une expression XPATH définie dynamiquement.

Le script que nous allons réaliser se divise comme précédemment en deux parties. La première partie présente un formulaire HTML destiné à récupérer un mot-clé. La seconde partie, sur base du mot-clé saisi par l'utilisateur, fouille le document XML.

Script appelé par le navigateur ('search.php') - première partie

```
1  - <?php
2  - # fichiers inclus
3  - require('html_tools.php');
4  -
5  - # fonction de mise en page (entête)
6  - print_header_page();
7  -
8  - # teste si un mot-clé a déjà été saisi
9  - # sinon : affichage d'un formulaire
10 - if (empty($_GET[keyword]))
11 - {
12 - echo "
13 - <form method=\"get\" action=\"\">
14 - <span >Mot-clé : </span>
15 - <input type=\"text\" size=\"10\" name=\"keyword\">
16 - <input type=\"submit\" value=\"ok\">
17 - </form>
18 - ";
19 - }
20 - else
21 - {
22 - # seconde partie
23 - }
24 -
25 - ?>
```

Ce script commence par inclure un fichier php servant de boîtes à outils 'html_tools.php'. Il appelle la fonction 'print_header_page()' dont la définition se trouve dans 'html_tools.php'. Cette fonction génère le code HTML d'entête de page.
Ensuite un test est effetué : si la variable globale '$_GET[keyword]' n'a pas été affectée (en d'autres termes si l'utilisateur n'a pas encore saisi de mot-clé) alors le script affiche un formulaire d'encodage HTML. Ce formulaire est des plus simple puisqu'il ne présente qu'un seul champ de saisie.

Script appelé par le navigateur ('search.php') - seconde partie

```
1  - <?php
2  - if (empty($_GET[keyword]))
3  - {
4  - # première partie
5  - }
6  - else
7  - {
8  - # parsing du document XML
```

```
9  - $xml_file = "D:\xml\biblio.xml";
10 - if(!$doc = domxml_open_file($xml_file))
11 -    {
12 -    die("Le document XML contient des erreurs !");
13 -    }
14 -
15 - # create new context
16 - $ctx = $doc->xpath_new_context();
17 -
18 - # récupération de noeuds avec XPATH
19 - $xpath_expr = "/bibliographie/livres/livre[contains(titre,
   \"$_GET[keyword]\")]/titre";
20 - $nodelist = $ctx->xpath_eval($xpath_expr);
21 -
22 - if (is_array($nodelist->nodeset))
23 -    {
24 -    # affichage du titre
25 -    foreach ($nodelist->nodeset as $node)
26 -       {
27 -       $textnode = $node->first_child();
28 -       print utf8_decode($textnode->node_value()) . "<br>";
29 -       }
30 -    }
31 - else
32 -    {
33 -    echo "Nous n'avons pas pu trouver de réponse
34 -    satisfaisante pour votre recherche sur \"$_GET[keyword]\"";
35 -    }
36 - }
37 -
38 - print_footer_page();
39 - ?>
```

Cette partie du script est activée si l'utilisateur est passé par le formulaire HTML.
Le script crée un objet DomDocument sur base du document XML fourni. Si le document n'est pas bien formé, un message d'erreur est affiché et le programme est interrompu.
Après avoir créé un contexte XPATH avec l'instruction 'xpath_new_context', l'expression XPATH est construite en y intégrant le mot-clé récupéré du formualire HTML. Cette expression est ensuite passée à l'instruction 'xpath_eval' pour traitement.
Si un résultat est obtenu (is_array), tous les noeuds récupérés sont passés en revue et traités. Dans le cas contraire un message d'avertissement est affiché à l'écran.

Manipulation DOM

Suppression d'un ensemble de noeuds

Il est possible de supprimer un noeud enfant quelconque, grâce à la méthode 'removeChild' de l'API DOM de PHP5. Par exemple :

Suppression d'un noeud

```
1  - <?php
2  - # Chargement du document XML
3  - $dom = new DOMDocument;
4  - $dom->load('biblio.xml');
5  -
6  - # Récupération de l'élément racine du document
7  - $root = $dom->documentElement;
8  -
9  - # Récupération et suppression du premier élément 'livre'
10 - $livre = $root->getElementsByTagName('livre')->item(0);
11 - $deletedLivre = $root->removeChild($livre);
12 -
13 - # Affichage du résultat
14 - echo $dom->saveXML();
15 - ?>
```

La méthode 'item' de l'objet DOMNodeList retourne un noeud spécifié par son index (en l'occurrence le premier élément). Une fois récupéré, cet élément est effacé de l'arborescence DOM via la méthode 'removeChild' de l'objet 'DOMNode'.

C'est un petit peu plus compliqué lorsqu'il s'agit d'effacer un groupe de noeuds sur base d'un critère de sélection - par exemple : effacer les noeuds vides. Prenons pour source le document XML suivant :

Fichier XML source - biblio.xml

```
1  - <?xml version="1.0" encoding="ISO-8859-1"?>
2  - <bibliographie>
3  - <livre/>
4  - <livre/>
5  - <livre/>
6  - <livre>
7  - <titre>Clandestin</titre>
8  - <auteur>James Ellroy</auteur>
9  - <editeur>Rivages</editeur>
10 - <lieu>Paris</lieu>
11 - <date>1990</date>
12 - <pages>445</pages>
13 - </livre>
14 - <livre/>
15 - <livre/>
16 - <livre>
17 - <titre>Le grand nulle part</titre>
18 - <auteur>James Ellroy</auteur>
19 - <editeur>Rivages</editeur>
20 - <lieu>Paris</lieu>
21 - <date>1990</date>
22 - <pages>445</pages>
23 - </livre>
24 - <livre/>
25 - </bibliographie>
```

Ce document contient un certain nombre d'éléments 'livre' dont plusieurs sont vides (<livre/>). Ce sont ces-derniers que nous allons essayer de supprimer du document xml.

Script exploitant l'interface DOM de PHP5

```
1  - <?php
2  - $dom = new DOMDocument;
3  - $dom->Load('biblio.xml');
4  - # Récupération d'une nodelist contenant tous les noeuds 'livre'
5  - $livres = $dom->getElementsByTagName('livre');
6  -
7  - $childNodes = array();
8  - # Transfert de la nodelist vers un tableau
9  - foreach($livres as $livre)
10 -    {
11 -    # Copie des noeuds dans un tableau
12 -    $childNodes[] = $livre;
13 -    }
14 -
15 - # Passage en revue des objets DomNode du tableau
16 - foreach ($childNodes as $childNode)
17 -    {
18 -    # Teste si l'élément possède des noeuds enfants
19 -    if (!$childNode->hasChildNodes())
20 -       {
21 -       # Suppression de l'élément dans l'objet DomDocument
22 -       $childNode->parentNode->removeChild($childNode);
23 -       }
24 -    }
25 -
26 - # Affichage du nouvel arbre DOM
27 - print $dom->saveXML();
28 - # Sauvergarde du nouvel arbre DOM sur disque
29 - $dom->save("biblio.xml");
30 - ?>
```

L'ensemble des éléments 'livre' sont récupérés dans une 'NodeList' via la méthode 'getElementsByTagName'. Les différents noeuds de la NodeList sont ensuite stockés dans un tableau (array) - ceci afin de pouvoir les parcourir/manipuler indépendamment de la mise à jour dynamique de l'arborescence DOM et ainsi éviter les interférences au niveau des identificateurs d'objets.

Manipulation DOM

Exploitation d'un canal RSS

Le format RSS (Real Simple Syndication) permet de décrire de façon succinte et standardisée le contenu d'un site. Sous forme de fichiers ou de flux dynamiques, cette information est mise à disposition sur internet par les diffuseurs de contenu (BBC, AFP, Apple ...) et est récupérée (à l'aide de logiciels syndicataires ou de scripts) par toute personne désirant l'exploiter dans un contexte particulier. A partir du moment où un site est mis à jour régulièrement et intensivement, il peut largement bénéficier du format RSS.

Ce format s'appuie sur la technologie XML. La description d'un site ou d'un contenu est en effet formalisée dans une structure balisée répondant à un standard. Du fait de sa simplicité, ce format a connu un rapide succés qui fait de lui l'une des applications XML les plus couramment utilisées.

PHP et son extension DOM XML offrent la possibilité d'exploiter des flux RSS : c'est-à-dire les récupérer, les parcourir, et les interpréter afin de les diffuser.

Script récupérant en DOM les titres d'un flux RSS

```
1  - <?php
2  - require('../design/html_tools.inc.php');
3  -
4  - print_header_page();
5  - $xml_file="http://www.alistapart.com/articles.rdf";
6  - $doc = domxml_open_file($xml_file);
7  - $ctx = $doc->xpath_new_context();
8  - $xpath_expr = "/rss/channel/item/title";
9  -
10 - $nodelist = $ctx->xpath_eval($xpath_expr);
11 - if (is_array($nodelist->nodeset))
12 - {
13 - foreach ($nodelist->nodeset as $node)
14 - {
15 - $textnode = $node->first_child();
16 - print utf8_decode($textnode->node_value()) . "<br>";
17 - }
18 - }
19 -
20 - print_footer_page();
21 - ?>
```

L'url du flux RSS est "http://www.alistapart.com/articles.rdf".
Les titres de ce flux sont repérés et recupérés grâce à une expression XPATH (/rss/channel/item/title).

Script récupérant en DOM les titres et les descriptions RSS

```
1  - <?php
2  - require('../design/html_tools.inc.php');
3  -
4  - print_header_page();
5  - $xml_file = "http://webnotes.ulb.ac.be/&rss=last";
6  - $doc = domxml_open_file($xml_file);
7  - $ctx = $doc->xpath_new_context();
8  - $xpath_expr = "/rss/channel/item";
```

```
9  -
10 - $nodelist = $ctx->xpath_eval($xpath_expr);
11 - if (is_array($nodelist->nodeset))
12 - {
13 - foreach ($nodelist->nodeset as $node)
14 - {
15 - echo '<p>';
16 - $titles = $node->get_elements_by_tagname("title");
17 - echo '<b>' . $titles[0]->get_content() . '</b><br>';
18 - $des = $node->get_elements_by_tagname("description");
19 - echo $des[0]->get_content() . '<br>';
20 - echo '</p>';
21 - }
22 - }
23 -
24 - print_footer_page();
25 - ?>
```

L'url du flux RSS est "http://webnotes.ulb.ac.be/&rss=last".
Les titres et les descriptions de ce flux sont recupérés grâce à l'instruction 'get_elements_by_tagname'.

Script traitant un flux RSS via une feuille XSLT

```
1  - <?php
2  - require('../design/html_tools.inc.php');
3  -
4  - print_header_page();
5  -
6  - $rss="http://nslog.com/index.rss";
7  - $doc = domxml_open_file($rss);
8  - $xslt=domxml_xslt_stylesheet_file(realpath("rss.xsl"));
9  - $result = $xslt->process($doc);
10 - print $xslt->result_dump_mem($result);
11 -
12 - print_footer_page();
13 - ?>
```

L'url du flux RSS est "http://nslog.com/index.rss".
C'est une feuille de style (en l'occurence 'rss.xslt') qui "habille" les informations présentes dans le flux RSS.

Feuille 'rss.xsl'

```
1  - <?xml version="1.0" encoding="iso-8859-1" ?>
2  - <xsl:stylesheet version="1.0"
3  - xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  - <xsl:output method="html" encoding="iso-8859-1"/>
5  -
6  - <xsl:template match="/rss">
7  - <xsl:apply-templates select="channel/item"/>
8  - </xsl:template>
9  -
10 - <xsl:template match="item">
11 - <p>
12 - <xsl:apply-templates select="title"/>
13 - <xsl:apply-templates select="description"/>
14 - </p>
15 - </xsl:template>
16 -
17 - <xsl:template match="title">
18 - <b><xsl:value-of select="."/></b><br/>
19 - </xsl:template>
20 -
```

```
21 - <xsl:template match="description">
22 - <xsl:value-of select="."/>
23 - </xsl:template>
24 - </xsl:stylesheet>
```

Feuille de style traitant le contenu d'un flux RSS (récupération et affichage des titres et de leur description).

Manipulation SAX

Parcours d'une structure XML - exploitation des entités externes

En XML, les entités externes rendent possible la construction d'un document XML à partir de plusieurs autres documents XML complémentaires. Cela permet notamment de segmenter un gros document en plusieurs sous-documents.

La déclaration d'une entité externe est précisée au niveau de la DTD et s'effectue comme suit :

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

Les références d'entités externes sont ensuite invoquées dans le document XML de la manière suivante :

```
&nom_entité;
```

Pour que PHP (et l'API SAX) puisse "suivre" ces références externes et donc en intégrer complètement le contenu dans l'analyse de la structure XML, il faut définir un gestionnaire XML de références externes via la fonction "xml_set_external_entity_ref_handler()". Voyons cela plus concrètement :

Fichier XML principal - info.xml

```
1  - <?xml version="1.0" encoding="iso-8859-1"?>
2  - <!DOCTYPE info [
3  - <!ENTITY news SYSTEM "listnews.xml">
4  - ]>
5  - <info>
6  - <items>
7  - <item>
8  - <title>Islams et mondialisation</title>
9  - <description>Cinquième conférence du cycle de regards sur les
   islams du nouveau siècle, " Islam et mondialisation, dans sa
   globalité " clôturera la série en compagnie d'Ali Serghini, le 8
   décembre à 20h, à la Maison des anciens.</description>
10 - </item>
11 - <item>
12 - <title>Jusqu'où va l'hérédité ?</title>
13 - <description>Jusqu'où va l'hérédité? Les yeux de maman, le nez de
   papa, bien sûr. Et... la dépression de maman, la violence de papa
   aussi? Dans nos comportements et nos traits de personnalité, quelle
   est la part de la génétique, quelle est la part de l'éducation ou de
   l'environnement?</description>
14 - </item>
15 - </items>
16 - &news;
17 - </info>
```

Dans ce document, on spécifie, lors de la déclaration de l'entité (ligne 3), une URI vers le fichier dont le contenu sera utilisé comme donnée de substitution.

Lors de l'appel de l'entité dans le document (ligne 16), l'ensemble des données référencées par l'URI spécifiée seront insérées à la place de la chaîne.

Fichier XML secondaire - listnews.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<items>
<item>
<title>La femme et la santé</title>
<description>La Coordination laïque de l'action sociale et de la santé et le Comité d'éthique du CHU Brugmann organisent une journée d'étude et de réflexion intitulée "La femme et la santé?".</description>
</item>
<item>
<title>Stratégie de recherche en bibliothèque</title>
<description>Un bureau de référence destiné à aider les étudiants avancés (mémorants, étudiants de 3e cycle, doctorants) et les chercheurs en sciences humaines est opérationnel à l'ULB.</description>
</item>
</items>
```

Dans le cas qui nous occupe, l'entité externe XML citée est tout simplement un document externe dont le contenu est du code XML.

Script php parcourant la structure XML - showinfo.php

```
<?php
# Variables globales
$listItems = array();
$countItems = 0;
$curTag = '';
$title = '';
$descr = '';

# Traitement lié au début de balise pour l'élément courant
function startElement($xp,$name,$attributes)
{
global $countItems, $curTag, $title, $description;
$curTag = "$name";
if ($name == 'ITEM')
   {
   $countItems++;
   $title = '';
   $descr = '';
   }
}

# Traitement lié à la fin de balise pour l'élément courant
function endElement($xp,$name)
{
global $title, $description, $listItems;
if ($name == 'ITEM')
   {
   # Enregistrement des données pertinentes
   $record['title'] = $title;
   $record['description'] = $descr;
   $listItems[] = $record;
   }
}

# Traitement des données caractères de l'élément courant
function characterDataHandler($xp,$data)
{
global $curTag, $title, $descr;
# Récupération des éléments ciblés
# Identification par rapport à l'élément courant
switch($curTag)
   {
   case 'TITLE':
      $title .= $data;
      break;
   case 'DESCRIPTION':
      $descr .= $data;
   break;
   }
}
```

```
51 -
52 - # Traitement lié aux entités XML externes
53 - function exEHandler($parser, $names, $base, $sysId, $pubId)
54 - {
55 - if ($sysId)
56 -    {
57 -    # Lancement du parsing pour une entitée
58 -    parse($sysId);
59 -    return true;
60 -    }
61 - else
62 -    {
63 -    return false;
64 -    }
65 - }
66 -
67 - # Fonction dédiée à l'activation d'une analyse XML
68 - function parse($xml_file)
69 - {
70 - # Création d'un analyseur XML
71 - $xp = xml_parser_create();
72 - # Affectation des gestionnaires d'événements
73 - xml_set_element_handler($xp, 'startElement', 'endElement');
74 - xml_set_character_data_handler($xp, 'characterDataHandler');
75 - xml_set_external_entity_ref_handler($xp, 'exEHandler');
76 - # Prise en compte d'un fichier source
77 - if (!($fp = fopen($xml_file, "r"))) die("I/O error: $xml_file");
78 - while ($data = fread($fp, 4096))
79 -    {
80 -    # Analyse d'un document XML
81 -    xml_parse($xp, $data)
82 -    }
83 - # Libération de la mémoire utilisée par le parseur
84 - xml_parser_free($xp);
85 - }
86 -
87 - # Lancement du programme
88 - parse('info.xml');
89 - # Affichage des informations récupérées
90 - foreach ($listItems as $item)
91 -    {
92 -    echo "<h2>Article :</h2>\n";
93 -    foreach ($item as $key => $value)
94 -       {
95 -       echo "<b>$key</b> : $value <br />";
96 -       }
97 -    echo "<br/><br/>\n\n";
98 -    }
99 - ?>
```

Ce script lance l'analyse SAX/XML (ligne 88) pour le document 'info.xml'. Les différentes fonctions précisées via les gestionnaires d'événements (lignes 73-75) sont chargées de récupérer le contenu des articles (titre et description) (lignes 43-48, 29-31).

Grâce à la mention du gestionnaire d'entités externes (ligne 75), l'analyse et donc la récupération des informations peuvent se faire même dans les documents XML complémentaires.

Résultat obtenu après analyse SAX

```
1  - <h2>Article :</h2>
2  - <b>title</b> : Islams et mondialisation<br />
3  - <b>description</b> : Cinquième conférence du cycle de regards sur
   les islams du nouveau siècle, " Islam et mondialisation, dans sa
   globalité " clôturera la série en compagnie d'Ali Serghini, le 8
   décembre à 20h, à la Maison des anciens.<br />
4  - --------------------------<br /><br />
5  -
6  - <h2>Article :</h2>
7  - <b>title</b> : Jusqu'où va l'hérédité ?<br />
8  - <b>description</b> : Jusqu'où va l'hérédité? Les yeux de maman, le
   nez de papa, bien sûr. Et... la dépression de maman, la violence de
```

```
   papa aussi? Dans nos comportements et nos traits de personnalité,
   quelle est la part de la génétique, quelle est la part de
   l'éducation ou de l'environnement?<br />
9  - --------------------------<br /><br />
10 -
11 - <h2>Article :</h2>
12 - <b>title</b> : La femme et la santé<br />
13 - <b>description</b> : La Coordination laïque de l'action sociale et
   de la santé et le Comité d'éthique du CHU Brugmann organisent une
   journée d'étude et de réflexion intitulée "La femme et la
   santé?".<br />
14 - --------------------------<br /><br />
15 -
16 - <h2>Article :</h2>
17 - <b>title</b> : Stratégie de recherche en bibliothèque<br />
18 - <b>description</b> : Un bureau de référence destiné à aider les
   étudiants avancés (mémorants, étudiants de 3e cycle, doctorants) et
   les chercheurs en sciences humaines est opérationnel à l'ULB.<br />
19 - --------------------------<br /><br />
```

Nous obtenons donc bien ici une recomposition d'informations à partir de différentes sources : les deux premiers articles sont issus du fichier XML principal (info.xml) et les deux derniers proviennent du fichier XML complémentaire (listnews.xml).