Capítulo 16: XML

Una de las tecnologías más necesarias hoy en dia y en el futuro en la mayoría de los portales de Internet que manejan información es XML. XML basado en SGML al igual que HTML es un lenguaje que permite definir otros lenguajes de tipo Mark-UP como por ejemplo HTML. El objetivo de XML es proveer a la Web de un lenguaje standard fuertemente estructurado que permita estructurar contenidos. En XML los documentos pueden ser validados contra un "DTD" (document type definition) para verificar si cumplen lo que dicho DTD determina. El intercambio de DTDs entre distintas instituciones es un buen medio de intercambiar informacion y poder validarla.

PHP provee funciones que permiten parsear documentos XML mediante la biblioteca "Expat" que puede conseguirse libremente en caso de ser necesario, aquellos que usen Apache como Web Server ya tienen expat en forma default.

PHP provee mediante expat un parser XML que puede configurarse para usar "handlers" específicos que manejen los distintos tipos de tags XML, para configurar handlers para los distintos tipos de tags se utilizan las siguientes funciones:

Supported XML handlers

PHP function to set handler	Event description
xml_set_element_handler()	Un evento de elemento se es levantado por expat cada vez que se encuentra un tag de apertura o cierre. Esta función permite setear handlers que se ejecuten cuando ocurren dichos eventos.
xml_set_character_data_handler()	Todo el contenido que no sea de markup de un xml incluyendo espacios en blanco y saltos de línea entre tags es considerado character_data. Es responsabilidad de la aplicación tratar estos datos y para ello es posible setear un handler que se invoque cada vez que se encuentren estos datos.
xml_set_processing_instruction_handler()	XM soporta el uso de "processing instructions" por ejemplo <?php?> es una, cada vez que el parser expat encuentra una PI se invoca esl handler seteado por esta función. Las PIs de tipo <?XML?> no disparan eventos porque no los maneja la aplicación de acuerdo al standard XML.
xml_set_default_handler()	Todo aquello que no corresponda a ningún otro handler disparará un evento deafult que puede capturarse con esta función.
xml_set_unparsed_entity_decl_handler()	Este handler se llama cuando aparecen declaraciones de entidades no parseables (NDATA).
xml_set_notation_decl_handler()	Handler que se invoca cuando se declara una notación XML.
xml_set_external_entity_ref_handler()	Esta función permite setear un handler a invocar cuando el parser xml encuentra una entidad parseable en forma externa con referencia a un file o URL

Notar que las funciones que describimos arriba no son los handlers sino las funciones que permiten definir los handlers a asociar con cada eventos (son funciones de binding)

XML Handlers:

Element Handlers:

xml_set_element_handler (parser_handler, string startElementHandler, string endElementHandler)

La función xml_set_element_handler recibe como parámetros un parser_xml (creado con xml_parser_create) y el nombre de dos funciones que serán los handlers a aplicar cuando se encuentren tags XML, la función necesita el nombre de dos funciones, una para cuando el tag xml abre y otra para cuando el tag xml cierra.

La función startElementHandler debe tener el siguiente prototipo:

boolean=element_start_handler(parser_handler,string_tag,attribs)
boolean=element_end_handler(parser_handler,string_tag)

La función recibe el tag en cuestión y si es un tag que abre recibe los parámetros del tag en caso de estar presentes en un vector asociativo como tercer parámetro. Un ejemplo de tag con atributos es:

<INFO code=”1234” name=”foo”>

En cuyo caso el nombre del tag es “INFO” y además se recibe un asociativo de la forma:
(“code”=>”1234”, “name”=>” foo”)

Como es de esperar los handlers para tags que cierran no reciben atributos.

Se supone que las funciones devuelve true si el parser_handler corresponde a un parser y false en caso contrario.

Character Data Handlers:

xml_set_character_data_handler (parser_handler, string Handler)

La función debe tener como prototipo:

boolean=charhandler(parser_handler, string_data)

Se supone que la función devuelve true si el parser_handler corresponde a un parser y false en caso contrario.

Processing Instructions Handlers:

xml_set_processing_instruction_handler (parser_handler, string_handler_name)

Un processing instruction en XML tiene la forma:

<?target data?>

Ejemplo <?php codigo?> Donde el target es PHP y el data es el código a ejecutar.

El handler responde al prototipo:

handler (parser_hanlder, string_target, string_data)

Default Handler:

xml_set_default_handler (parser_handler, string_handler_name)

Y el handler es de la forma:

handler (parser_hanlder, string_data)

Creación de un Parser XML

Para crear un parser xml se deb usar:

xml_handler=xml_parser_create();

Que crea un parser xml expat y devuelve un handler al mismo.

Parsing:

La función de parsing se invoca usando:

boolean=xml_parse(parser_handler, string_data, boolean);

La función recibe un handler a un parser creado y datos a parsear, se puede llamar a la función “n” veces con distintos pedazos de datos para que la función vaya parseando el documento de a partes.
El tercer parámetro es opcional , si es true indica que los datos que se pasan son los últimos a parsear.
Devuelve true si no hay problemas y false si se produce un error.

Una vez que se termina de parsear un documento es necesario desalocar el parser instanciado usando:

xml_parser_free(parser_hadler);

Uso de Clases:

A veces es conveniente definir que los handlers de un parser_xml sean métodos de una clase o bien crear una clase para parsear determinado tipo de documento XML. Tal vez sea una buena idea definir una clase base para parsear XML en general y luego clases derivadas para los distintos tipos de documentos XML que manejamos. Los nombres que se pasan a las funciones que setean los handlers no prevén construcciones usando objetos por lo que se creo una función especial xml_set_object que le indica al parser xml que los nombres de los handlers registrados son métodos de un objeto determinado:

xml_set_object(xml_handler, referencia_a_un_objeto);

Por ejemplo si estamos dentro de una clase que parsea xml:

xml_set_object($this->xml_parser, &$this);

Ejemplo 1:

Veamos un ejemplo sobre como usar estos handlers para setear manejadores para tags XML, como crear un parser de XML y como parsear el documento, en el ejemplo convertimos XML en HTML:

```
 1
 2 $file = "data.xml";
 3 $map_array = array(
 4     "BOLD"     => "B",
 5     "EMPHASIS" => "I",
 6     "LITERAL"  => "TT"
 7 );
 8
 9 function startElement($parser, $name, $attrs) {
10     global $map_array;
11     if ($htmltag = $map_array[$name]) {
12         print "<$htmltag>";
```

```
13     }
14 }
15
16 function endElement($parser, $name) {
17     global $map_array;
18     if ($htmltag = $map_array[$name]) {
19         print "</$htmltag>";
20     }
21 }
22
23 function characterData($parser, $data) {
24     print $data;
25 }
26
27 $xml_parser = xml_parser_create();
28 // use case-folding so we are sure to find the tag in $map_array
29 xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
30 xml_set_element_handler($xml_parser, "startElement", "endElement");
31 xml_set_character_data_handler($xml_parser, "characterData");
32 if (!($fp = fopen($file, "r"))) {
33     die("could not open XML input");
34 }
35
36 while ($data = fread($fp, 4096)) {
37     if (!xml_parse($xml_parser, $data, feof($fp))) {
38         die(sprintf("XML error: %s at line %d",
39                     xml_error_string(xml_get_error_code($xml_parser)),
40                     xml_get_current_line_number($xml_parser)));
41     }
42 }
43 xml_parser_free($xml_parser);
44
```

Si el documento XML fuera:

Example 5. xmltest2.xml

```
 1
 2 <?xml version="1.0"?>
 3 <foo>
 4 <bold>Este es el titulo</bold>
 6 <emphasis>Descubren sopa que cura la gripe</emphasis>
 7 <literal>Una nueva sopa de <bold>cebolla</bold> que curaria
 8 la gripe fue descubierta hoy en la ciudad de Bolivar, provincia
 9 de Buenos Aires.</literal>
10 </foo>
11
```

El Browser recibiria el siguiente documento:

Example 5. xmltest2.xml

```
  1
  2
  3 <b>Este es el titulo</b>
  6 <i>Descubren sopa que cura la gripe</i>
  7 <tt>Una nueva sopa de <b>cebolla</b> que curaria
  8 la gripe fue descubierta hoy en la ciudad de Bolivar, provincia
  9 de Buenos Aires.</tt>
 10 </foo>
 11
```

Ejemplo 2:

En el siguiente ejemplo vamos a ver una clase definida para parsear noticias en XML en un cierto formato predefinido y mostrarlas en una página convirtiéndolas en HTML, vamos a ver como se usa Orientación a Objetos para encapsular el parser en una sola clase:

El formato de documentos XML que parsea la clase es:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE NewsBoy SYSTEM "NewsBoy.dtd">

<NewsBoy>

<story>
        <date>03/31/2000</date>
        <slug>Sooo Busy !</slug>
        <text>
        I haven't posted anything here for a while now as I have been
busy with work (have to pay those bills!). <newline></newline>
        I have just finished a neat little script that stores a complete
record set in a session variable after <newline></newline>
        doing an SQL query. The neat part is that an XML doc is stored in
the session variable an when paging <newline></newline>
        through the results (often near 1000!) the script displays 50
results at a time from the XML doc in the <newline></newline>
        session variable instead of doing another query against the
database. It takes a BIG load off of the <newline></newline>
        database server.
        </text>
</story>

<story>
        <date>03/25/2000</date>
        <slug>NewsBoy Class</slug>
        <text>
        Converted Newsboy to a PHP class to allow better abstraction (as
far as PHP allows.)   <newline></newline>
        Guess that means this is version 0.02 ?!
                        <newline></newline>
        Newsboy will have a section of it's own soon on how to use and
customize the class.    <newline></newline>
        </text>
```

```
</story>

<story>
        <date>03/24/2000</date>
        <slug>NewsBoy is up!</slug>
        <text>
        I have just finished NewsBoy v0.01 !!! <newline></newline>
        It looks quite promising. You may ask, &quot;What the heck is
it?!&quot;. <newline></newline>
        Well it's a simple news system for web-sites, written in PHP,
that makes use of XML for <newline></newline>
        the news data format allowing easy updating and portability
between platforms.
        It uses the built in expat parser for Apache.
        This is just the very first version and there will be loads of
improvements as the
        project progresses.
        </text>
</story>

<story>
        <date>03/24/2000</date>
        <slug>Romeo must Die</slug>
        <text>
        Saw a really cool movie today at Mann called 'Romeo must Die'
<newline></newline>
        Nice fight scenes for a typical kung-fu movie with some 'Matrix'
style effects. <newline></newline>
        One particular cool effect was the 'X-Ray Vision' effect that
occured in various fight scenes. <newline></newline>
        The hero, played by Jet Li, strikes a bad guy and you can see the
bone in his arm crack, in X-RAY vision. <newline></newline>
        There were some funny scenes too when Jet has to play American
football with the bad guys. <newline></newline>
        The official website for the movie is &lt;A
HREF='http://www.romeo-must-die.com'&gt; here &lt;/A&gt;
<newline></newline>
        <newline></newline>
        </text>
        &lt;IMG
SRC=&quot;http://a1996.g.akamaitech.net/7/1996/25/e586077a88e7a4/romeomus
tdie.net/images/image15.jpg&quot; WIDTH=300 &gt;
</story>

</newsboy>
```

El código de la clase es:

```
<?php
/*
NewsBoy: a News system for the web written in PHP by Justin Grant
(Web: jusgrant.cjb.net or justin.host.za.net Mail: justin@glendale.net)
25 March V0.0.2 Converted Newsboy to a PHP class, allowing the layout to be easily modified.
Also added made the HTML that is genrated a little easier to read.
24 March V0.0.1 Just completed the intial version, very rough and basic.
*/

class newsboy {

var $xml_parser;
var $xml_file;
var $html;
var $open_tag ;
var $close_tag ;

//Class Constructor
function newsboy() {
$this->xml_parser = "";
$this->xml_file = "";
$this->html = "";
$this->open_tag = array( //these are the default settings but they are quite easy to modify
"NEWSBOY" => "\n<!-- XML Starts here -->\n<TABLE COLS=1 CELLPADDING=5>",
"STORY" => "<TR><TD BGCOLOR=#222222>",
"DATE" => "<FONT COLOR=#BBBB00>",
"SLUG" => "<FONT COLOR=#00AACC><B>  ",
"TEXT" => "<FONT COLOR=#AAAAAA>",
"PIC" => "",
"NEWLINE" => ""
);
$this->close_tag = array(
"NEWSBOY" => "</TABLE>\n<!-- XML Ends here -->\n\n",
"STORY" => "</TD></TR>",
"DATE" => "</FONT>",
"SLUG" => "</B></FONT><BR>",
"TEXT" => "</FONT>\n",
"PIC" => "<IMAGE SRC=",
"NEWLINE" => "<BR>"
);

}
//Class Destructor (has to be invoked manually as PHP does not support destructors)
function destroy() {
xml_parser_free($this->xml_parser);
}

//Class Members
function concat($str) {
$this->html .= $str;
}

function startElement($parser, $name, $attrs) {
//global $open_tag;
```

```
if ($format= $this->open_tag[$name]) {
$this->html .= $format;
}
}

function endElement($parser, $name) {
global $close_tag;

if ($format= $this->close_tag[$name]) {
$this->html .= $format;
}
}

function characterData($parser, $data) {
$this->html .= $data;
}

/*
function PIHandler($parser, $target, $data) {
//switch ( strtolower($target)){
// case "php":
eval($data);
// break;
//}
}
*/

function parse() {

$this->xml_parser = xml_parser_create();
xml_set_object($this->xml_parser, &$this );
// use case-folding so we are sure to find the tag in $map_array
xml_parser_set_option($this->xml_parser, XML_OPTION_CASE_FOLDING, true);
xml_set_element_handler($this->xml_parser, "startElement", "endElement" );
xml_set_character_data_handler($this->xml_parser, "characterData" );
//xml_set_processing_instruction_handler($this->xml_parser, "PIHandler");

if (!($fp = fopen($this->xml_file, "r"))) {
die( "could not open XML input");
}

while ($data = fread($fp, 4096)) {
if (!xml_parse($this->xml_parser, $data, feof($fp))) {
die(sprintf( "XML error: %s at line %d",
xml_error_string(xml_get_error_code($this->xml_parser)),
xml_get_current_line_number($this->xml_parser)));
}
}
}
}
?>
```

En primer lugar se definen los hanlders startElement y closeElement en donde se cuenta con un array asociativo que relaciona el tag xml con el código html a incluir, como vemos se verifica si existe el nombre del tag como clave del vector asociativo (if $clave=$vector[$clave]) si el tag existe entonces se recupera el código a incluir y se appendea al data_member "html" del objeto que será el código html resultante.

El character_data_handler simplemente agrega los datos que recibe al codigo de salida.

El método principal es "parse()", allí se crea un parser xml y se guarda el handler en $this->xml_parser, luego se usa xml_set_object para indicar que las funciones que registraremos como handlers son métodos de este objeto de la forma **xml_set_object($this->xml_parser, &$this)** luego se utiliza xml_parser_set_option para indicar que el parser es case-insensitive. Luego con xml_set_element_handler, xml_set_character_handler se registran los métodos a usar en el xml_parser.

Luego se parsea el file xml pasado al objeto usando una lectura simple de archivos en bloques de 4Kb, usando la función feof como tercer parametro de xml_parse para saber si quedan o no más datos en el archivo.

La forma de usar esta clase es muy sencilla:

$news = new newsboy();
$news->xml_file = "xml/mynews.xml";

o bien:

$news->xml_file = "http://xmldocs.mysite.com/mynews.xml"

Luego llamamos al método parse del objeto:

$news->parse();

Y tomamos el resultado y por ejemplo lo entregamos al browser:

print ($news->html);

Por ultimo destruimos el objeto:

$news->destroy();

Uso de PHP desde XML:

Usando el processing_instruction_handler podemos combinar incluir código PHP dentro de un documento XML, por ejemplo:

```
function processphp($parser,$target,$data) {
  if($target==”php”) {
    eval($data);
  }
}

xml_set_processing_instruction_handler($xml_parser,"processphp");
```

La función "eval" recibe un string con código php y lo ejecuta, como podemos ver con este esquema podemos tener un xml de la forma:

```
<?xml version="1.0"?>
<foo>
<titulo>Descubren sopa curadora en Catelar</titulo>
```

```
<copete>Una sopa con propiedades milagrosas fue descubierta hoy</copete>
<?php print("que cosa!");?>
<cuerpo>La sopa de cebollas del padre Enrico de la iglesia de Castelar
ademas de ser muy sabrosa tiene propiedades curadoras</cuerpo>
</foo>
```

Lo cual nos da opciones muy poderosas de definición y estructuración de contenidos usando XML más toda la posibilidad de programar usando php.

Uso de Coccon con PHP

XML es un lenguaje pensado para estructurar contenidos definiendo lenguajes de marcación, validar documentos y proveer un marco de intercambio de documentos y contenidos. Para poder usar XML desde un Web-Site es necesario definir alguna tecnología que permita transformar los contenidos definidos en XML en "presentación" típicamente usando HTML. Para transformar XML en html pueden usarse las funciones de parsing que estudiamos en php. Sin embargo existe un lenguaje creado específicamente para manjar transformaciones de documentos XML denominado XSL, XSL es un lenguaje de transformación de documentos XML en otros documentos XML, como es posible considerar HTML como un sub-set de XML es posible transformar XML en HTML usando XSL, mediante el uso de XSL es posible definir otras transformaciones para el mismo documento XML como por ejemplo un dispositivo WAP u otros.

Coccon es un proyecto de la Apache Software Foundation que provee un conjunto de Servlets que se integran con el Apache Web Server para maipular documentos XML, una de las cosas que Cocoon maneja es precisamente las transformaciones XSL.

La mejor estrategia de integración entre XSL y PHP mientras que PHP no soporte nativamente XSL es el uso de Cocoon, habitualmente se puede desde PHP generar un documento XML usando clases de generación de XML a partir de datos en la base de datos u otros formatos y luego generar un archivo XML, a continuación solo queda redireccionar al browser al archivo XML generado para que Coccon entre en acción parseando el documento XML y realizando la transformación que se desee usando la plantilla XSL seleccionada. Este marco de trabajo si bien es muy nuevo actualmente como para que podamos aportar datos sobre su funcionamiento puede ser uno de los caminos más acertados para integrar PHP, XML, XSL y HTML separando adecuadamente Aplicaciones, Contenidos y Presentación de los mismos.