Parsing XML with the DOM Extension for PHP 5[1]

by Octavia Andreea Anghel[2]

DOM (Document Object Model) is a W3C standard based on a set of interfaces, which can be used to represent an XML or HTML document as a tree of objects. A DOM tree defines the logical structure of documents and the way a document is accessed and manipulated. Using DOM, developers create and build XML or HTML documents, navigate their structures, and add, modify, or delete elements and content.

The DOM can be used with any programming language, but in this article we will use the DOM extension for PHP 5. This extension is part of the PHP core and doesn't need any installation.

The PHP object tree is built from nodes named according to XML. Some of the most familiar DOM nodes are:

- The document node, represented by the DOMDocument interface
- The element node, represented by the DOMElement interface
- The attribute node, represented by the DOMAttr interface
- The comment node, represented by the DOMComment interface
- The text node, represented by the DOMText interface

Extracting the PHP Tree Objects Associated with an XML Document

In this section, you will see how to extract some elements and their values from a PHP tree object. You will need the XML document Book.xml, which is listed below:

```
<?xml version="1.0? encoding="UTF-8? standalone="yes" ?>
<book>
<!-XML Processing [part I] ->
     <name>XML Processing I</name>
          <author>John Smith Jr.</author>
          <publisher>HisOwnTM</publisher>
          <ISBN>111-222-333-4441</ISBN>
          <contents>
               <chapter_I>
                    <title>What is XML about ?</title>
                    <content>XML (Extensible Markup Language) is a …
                    </content>
               </chapter_I>
               <chapter_II>
                    <title>SAX</title>
                    <content>SAX is a simple API for …</content>
```

[1] http://www.stableflow.com/articles/php/parsing-xml-with-the-dom-extension-for-php-5/ (Posted on November 8, 2010 by fes_aw)

[2] ***Octavia Andreea Anghel*** is a senior PHP developer currently working as a primary trainer for programming teams that participate at national and international software-development contests. She consults on developing educational projects at a national level. She is a coauthor of the book "XML Technologies: XML in Java" (Albastra, ISBN 978-973-650-210-1), for which she wrote the XML portions. In addition to PHP and XML, she's interested in software architecture, web services, UML, and high-performance unit tests. *Click here* to e-mail her.

```
            </chapter_II>
        <chapter_III>
            <title>StAX</title>
            <content>Much powerful and flexible, StAX, is very…
            </content>
        </chapter_III>
        <chapter_IV>
            <title>DOM
            <subtitle>DOM concept
                <continut>Starting to use DOM…</continut>
            </subtitle>
            <subchapter_IV_I>
                <title>First DOM application…</title>
                <content>Here it is your first DOM application…
                </content>
            </subchapter_IV_I>
            </title>
        </chapter_IV>
        <end>The end…</end>
        </contents>
<!- See you in XML Processing [part II] ->
</book>
```

Save this document into the same directory as the associated PHP applications.

The below application extracts the associated tree objects of Book.xml and displays the first occurrences of child nodes using the `getElementsByTagName` method, which belongs to the `DOMNodeList` `DOMElement::getElementsByTagName (string $name)` returns all descendants with the given $name tag name.

```
<?php
// Create a document instance
$doc = new DOMDocument();

//Load the Book.xml file
$doc->load( 'Book.xml' );

//Searches for all elements with the "book" tag name
$books = $doc->getElementsByTagName( "book" );

//Searches for all elements with the "author" tag name
$authors = $doc->getElementsByTagName( "author" );

//Returns the first element found having the tag name "author"
$author = $authors->item(0)->nodeValue;

//Searches for all elements with the "publisher" tag name
$publishers = $doc->getElementsByTagName( "publisher" );

//Returns the first element found having the tag name "publisher"
$publisher = $publishers->item(0)->nodeValue;

//Searches for all elements with the "name" tag name
$titles = $doc->getElementsByTagName( "name" );
```

```
//Returns the first element found having the tag name "name"
$title = $titles->item(0)->nodeValue;

//Printing the found values
echo "$title - $author - $publisher n";
?>
```

The listing result is:

```
XML Processing I - John Smith Jr. - HisOwnTM
```

Browse Through a Recursive PHP Object tree

The function below can be used to browse through all the nodes of PHP tree objects and to browse recursively the whole subtree, which has as a root the node taken as argument ($node) and the list name and values of each encountered node.

```
function getNodesInfo($node) {
     if ($node->hasChildNodes()) {
          $subNodes = $node->childNodes;
          foreach ($subNodes as $subNode) {
             if (($subNode->nodeType != 3) ||
                (($subNode->nodeType == 3) &&
                 (strlen(trim($subNode->wholeText))>=1))) {
                 echo "Node name: ".$subNode->nodeName."n";
                 echo "Node value: ".$subNode->nodeValue."n";
             }
             getNodesInfo($subNode);
           }
     }
}
```

Note: Notice that the empty text nodes were removed for a better view of the outcome using this condition line:

```
if (($subNode->nodeType != 3) ||
   (($subNode->nodeType == 3) &&
   (strlen(trim($subNode->wholeText))>=1)))
```

The alternative for this condition is the predefined preserveWhiteSpace function, which removes redundant white space and has the default set to TRUE.

The next application uses Book.xml and the recursive function getNodesInfo to list the entire object tree associated with the XML document:

```
<?php
//Create a document instance
$doc = new DOMDocument();

//Load the Book.xml file
$doc->load( 'Book.xml' );
```

```
//Setting the object tree root
$root = $doc->firstChild;

//The recursive function that list all the nodes of the tree
function getNodesInfo($node) {
     if ($node->hasChildNodes()) {
        $subNodes = $node->childNodes;
        foreach ($subNodes as $subNode) {
           if (($subNode->nodeType != 3) ||
              (($subNode->nodeType == 3) &&
              (strlen(trim($subNode->wholeText))>=1))) {
              echo "Node name: ".$subNode->nodeName."\n";
              echo "Node value: ".$subNode->nodeValue."\n";
            }
        getNodesInfo($subNode);
        }
     }
}

//The getNodesInfo function call
getNodesInfo($root);
?>
```

Note: Notice that the empty text nodes were removed for a better view of the outcome using this condition line:

```
if (($subNode->nodeType != 3) ||
   (($subNode->nodeType == 3) &&
    (strlen(trim($subNode->wholeText))>=1)))
```

The alternative for this condition is the predefined preserveWhiteSpace function, which removes redundant white space and has the default set to TRUE.

Adding New Nodes

To add a new node in the object tree, you can use one of the two methods of the DOMNode interface, `appendChild` and `insertBefore`. The difference between them is that the `appendChild` method will insert a new child at the end of the XML document while the `insertBefore` method will insert a new child before a specified node.

The following are the prototypes of those two methods.

1. `DOMNode DOMNode::appendChild (DOMNode $newnode)`: This function appends the `$newnode` argument at the end of an existing list of children or creates a new list of children. The child can be created using one of the two methods of the DOMDocument interface, `createElement` or `createTextNode`, as described below:

- `DOMElement createElement (string $name [, string $value ])`: This method creates an instance of the DOMElement class. The `$name` argument represents the tag name of the element and the $value argument represents the value of the element. The value can be set later using the DOMElement->nodeValue property.
- `DOMText createTextNode (string $content)`: This method creates an instance of the DOMText class. The $content argument represents the content of the new text node created.

The following example appends the <bibliography> node at the end of the object tree using the above function:

```
//Create a new element and add it to the root using the appendChild method
$newElement = $dom->createElement('bibliography','Martin Didier, Professional
XML');

//The appendNewChild function call
appendNewChild($root,$newElement);

//The function which will append a new child to the root
function appendNewChild($currentNode, $node)
{
//appending a new children to the root
$currentNode->appendChild($node);
}
```

2. DOMNode DOMNode::insertBefore (DOMNode $newnode [, DOMNode $refnode]):This method inserts a new node before the reference node. The `$node` argument represents the new node that will be inserted and the `$refnode` argument represents the reference node. If `$refnode` is missing, then the new node is appended to the children.

In this example the <foreword> child is added before the <publisher> child.

```
//add a new children foreword using the insertBefore method

<pre>$newElement = $dom->createElement('foreword','What I love about this
book is that it grew out of just such a process, and shows it on every
page.');</pre>

//Setting the reference node
$allContents = $dom->getElementsByTagName('publisher');
$contents = $allContents->item(0);

//The insertNewChild function call
insertNewChild($contents,$newElement);

//The function which will insert a new child to the first occurrence of the
$currentNode node

function insertNewChild($currentNode, $node)
{
$currentNode->insertBefore($node, $currentNode->firstChild);
}
```

Cloning of a Node

Cloning a node is the creation of a node that matches the current one. This can be done using the cloneNode method of the DOMNode interface, which has the following prototype:

DOMNode DOMNode::cloneNode ([bool $deep]): This method creates a clone of the current node; the $deep argument specifies whether the descendants of the current node will be copied too or not. The default value is FALSE.

The following function clones the <author> element and will append it to the object tree:

```
//Setting the reference node
$author = $root->getElementsByTagName('author')->item(0);

//The cloningNode function call
cloningNode($author);

//This function clone the $currentNode
function cloningNode($currentNode)
{
$clonenode = $currentNode -> cloneNode(true);
$newnode = $currentNode->appendChild($clonenode);
}
```

Removing Child Nodes

To remove a node from the object tree, you should use the removeChild method from the DOMNode interface. The removeChild method prototype is:

DOMNode DOMNode::removeChild (DOMNode $oldnode): This function removes a node's children from a list of children. The $oldnode argument represents the deleted node.

In the following example, the PHP function removes the <bibliography> child:

```
//Setting the reference node
$bibliography = $root->getElementsByTagName('bibliography')->item(0);

//The removingChild function call
removingChild($bibliography);

//This function remove the $currentNode node

function removingChild($currentNode)
{
$oldbibliography = $root->removeChild($currentNode);
}
```

Replacing Child Nodes

To replace an existing node with a new node, you can use the replaceChild method of the DOMNode interface. Its prototype is:

DOMNode DOMNode::replaceChild (DOMNode $newnode , DOMNode $oldnode): This function replaces the $oldnodechild with the $newnode child, only if the new node is not already a child.

For example, let's suppose that we want to replace the ISBN child name and value with the code child:

```
//Setting the reference node
$element = $dom->getElementsByTagName('ISBN')->item(0);

//Creating the new element node which will be replaced the $element node
$code = $dom->createElement('code', '909090');

//The replacingNode function call
replacingNode($code,$element);

//This function replace the $currentNode node with the $node node
function replacingNode($currentNode, $node)
{
$node->parentNode->replaceChild($currentNode, $node);
}
```

Import a Node from Another Tree Object

The method used to import a node from another tree object is `importNode` and its prototype is:

DOMNode DOMDocument::importNode (DOMNode $importedNode [, bool $deep]): This method allows importing the object tree associated with the current XML document of a node from a tree object associated with another XML document specified by the argument `$importedNode`. Because importing is done by copying, the method has specificity, i.e. the imported node represents a copy of the original node. This means that the import can be done without altering the trees objects from which they come. The `$deep` argument represents the manner in which the node will be imported. If it is true, then it will create a deep copy, which means that the whole subtree of the node will be imported. If the `$deep` argument is false, it will be created just for the node.

The next application (importeNode.php) imports the <continue> node from the Book_continue.xml into Book.xml.

The Book_continue.xml XML document is listed below:

```
<?xml version="1.0? encoding="UTF-8? standalone="yes" ?>
<!-chapter V->
<continue>
     <chapter_V>
          <title>XPath</title>
          <content>XPath is language for…</content>
     </chapter_V>
     <![CDATA[
     This chapter is a bonus to...
     ]]>
     <printing cap_I="click_here_for_chapter_I"
     cap_II="click_here_for_chapter_II"
     cap_III="click_here_for_chapter_III"
     cap_IV="click_here_for_chapter_IV"
     cap_V="click_here_for_chapter_V" />
</continue>
```

Here is the listing for importeNode.php:

```
<?php

$olddoc = new DOMDocument;
$olddoc->load("Book_continue.xml");

// The node we want to import to a new document
$node = $olddoc->getElementsByTagName("continue")->item(0);
$newdoc = new DOMDocument;
$newdoc->formatOutput = true;
$newdoc->load("Book.xml");

// Import the node, and all its children, to the document
$node = $newdoc->importNode($node, true);

// And then append it to the root node
$newdoc->documentElement->appendChild($node);

echo "nThe 'new document' after copying the nodes into it:n";
$root = $newdoc->firstChild;

function getNodesInfo($node)
{
     if ($node->hasChildNodes())
     {
          $subNodes = $node->childNodes;
          foreach ($subNodes as $subNode)
          {
          if (($subNode->nodeType != 3) ||
             (($subNode->nodeType == 3) &&
              (strlen(trim($subNode->wholeText))>=1)))
               {
                    echo "Node name: ".$subNode->nodeName."n";
                    echo "Node value: ".$subNode->nodeValue."n";
               }
          getNodesInfo($subNode);
          }
     }
}

getNodesInfo($root);

?>
```

Creating an Object Tree from Zero

The next application (Listing Flowers.php) creates an object tree and the XML document (Flowers.xml) associated to it using the previously described methods of the DOM extension for PHP 5:

```
<?php

//
$document = new DOMDocument();
$document->formatOutput = true;
```

```
//Create the <flowers> root element
$root = $document->createElement( 'flowers' );
$document->appendChild( $root );

//Create the <tulips> children of the root
$tulips = $document->createElement( 'tulips' );

//Create the first children of the <tulips> element,<bulbs>, and setting its
attribute
$bulbs_1 = $document->createElement( 'bulbs' );
$bulbs_1->setAttribute('price','€ 7.65');
$bulbs_1->appendChild($document->createTextNode( 'parrot'));
$tulips->appendChild( $bulbs_1 );

//Create the second children of the <tulips> element,<bulbs>, and setting its
attribute
$bulbs_2 = $document->createElement( 'bulbs' );
$bulbs_2->setAttribute('color','magenta');
$bulbs_2->appendChild($document->createTextNode( 'lily flowering' ));
$tulips->appendChild( $bulbs_2 );

//Append to the root the <tulips> children
$root->appendChild( $tulips );

//Save the object tree into the Flowers.xml XML document
echo $document->saveXML();
$document->save('Flowers.xml');

?>
```

The Flower.xml document looks like this:

```
<?xml version="1.0? encoding="ISO-8859-1??>
<flowers>
     <tulips>
          <bulbs price="€ 7.65?>parrot</bulbs>
          <bulbs color="magenta">lily flowering</bulbs>
     </tulips>
</flowers>
```

Conclusion

In this article, you have seen all the interfaces of the DOM extension for PHP 5 used in some basic examples and even how to create a new XML document from zero.

Original Article: http://www.phpbuilder.com/columns/DOM-XML-extension/Octavia_Anghel102710.php3