Handling XML Data in PHP5 using SimpleXML[1]

Introduction

XML, Extensible Markup Language, is a general-purpose markup language which can be used for storing arbitrary data in a structured way. It is often used for sharing data between applications and a common usage of XML is for instance RSS feeds.

With the emerge of PHP 5, support for handling XML data has greatly improved and in this tutorial we will take a look at the features in PHP 5 which we can use to parse, alter and create XML documents. At the end of the tutorial we will also take a look at how you can create an RSS feed without writing a single line of XML and then we will make it into a reusable class which you can implement in your own applications. Additionally we'll create a *really* simple RSS reader.

Basic knowledge of XML is a prerequisite for this tutorial. Knowledge of XPath would be a good idea as it will be used a bit in this tutorial, however, this is not entirely important. Furthermore, OOP knowledge is a requirement as well.

Right, let's get started...

Parsing XML

PHP 5 has a class called `SimpleXML` which is [relatively] simple to use.

Throughout this tutorial we will use the file `books.xml` which [the author] created using data from the top three books on Amazon.com's Editor's Picks: Best Books of 2007. The content of the file is:

```
<?xml version="1.0"?>
<books>
  <book isbn="978-1594489501">
    <title>A Thousand Splendid Suns</title>
    <author>Khaled Hosseini</author>
    <publisher>Riverhead Hardcover</publisher>
    <amazon_price>14.27</amazon_price>
  </book>
  <book isbn="978-1594489587">
    <title>The Brief Wondrous Life of Oscar Wao</title>
    <author>Junot Diaz</author>
    <publisher>Riverhead Hardcover</publisher>
    <amazon_price>14.97</amazon_price>
  </book>
  <book isbn="978-0545010221">
    <title>Harry Potter and the Deathly Hallows</title>
    <author>J. K. Rowling</author>
```

[1] Source: http://92798.net/article.php?id=28

```
    <publisher>Arthur A. Levine Books</publisher>
    <amazon_price>19.24</amazon_price>
  </book>
</books>
```

We can load our data into `SimpleXML` in two ways. Either we can pass a string to `SimpleXML` or we can tell it where the file is:

```
<?php
// Passing the XML
$data = file_get_contents('books.xml');
$books = SimpleXMLElement($data);
//--------------------
// Passing a filename
$books = SimpleXMLElement('books.xml', null, true);
?>
```

If you have the file stored you should obviously use the latter approach as there is no reason to get the file contents ourselves when we can get `SimpleXML` to do it for us. The first way could be used if you already have some XML data which could for instance be retrieved from some web service. I only used `file_get_contents()` for the purpose of that example.

The second argument is used to set certain options. We won't touch that in this tutorial, but you are free to research that yourself. The third argument specifies if the first argument is a filename (`true`) or actual XML data (`false`) and defaults to `false`. If `allow_url_fopen` is set to `true` in `php.ini` then you can specify a URL as well.

Another way of loading a file is by using the `simplexml_load_file()` function. In that case it will look like this:

```
$books = simplexml_load_file('books.xml');
```

Now let's turn our XML data into an HTML table:

```
<?php
// load SimpleXML
$books = new SimpleXMLElement('books.xml', null, true);

echo <<<EOF
<table>
        <tr>
                <th>Title</th>
                <th>Author</th>
                <th>Publisher</th>
                <th>Price at Amazon.com</th>
                <th>ISBN</th>
        </tr>

EOF;
foreach($books as $book) // loop through our books
{
        echo <<<EOF
        <tr>
```

```
                <td>{$book->title}</td>
                <td>{$book->author}</td>
                <td>{$book->publisher}</td>
                <td>\${$book->amazon_price}</td>
                <td>{$book['isbn']}</td>
        </tr>

EOF;
}
echo '</table>';
?>
```

This will result in a table looking like this:

```
<table>
        <tr>
                <th>Title</th>
                <th>Author</th>
                <th>Publisher</th>
                <th>Price at Amazon.com</th>
                <th>ISBN</th>
        </tr>
        <tr>
                <td>A Thousand Splendid Suns</td>
                <td>Khaled Hosseini</td>
                <td>Riverhead Hardcover</td>
                <td>$14.27</td>
                <td>978-1594489501</td>
        </tr>
        <tr>
                <td>The Brief Wondrous Life of Oscar Wao</td>
                <td>Junot Diaz</td>
                <td>Riverhead Hardcover</td>
                <td>$14.97</td>
                <td>978-1594489587</td>
        </tr>
        <tr>
                <td>Harry Potter and the Deathly Hallows</td>
                <td>J. K. Rowling</td>
                <td>Arthur A. Levine Books</td>
                <td>$19.24</td>
                <td>978-0545010221</td>
        </tr>
</table>
```

Right, so you say you wanted to display the title of the second book you'd do this (remember that array indices start from 0):

```
echo $books->book[1]->title;
```

The ISBN number of the same book is displayed like this:

```
echo $books->book[1]['isbn'];
```

XPath

You can also run XPath queries. They are run using the `SimpleXMLElement::xpath()` method that method takes a single argument, the XPath query. If we wanted to select all the book titles we could do something like this:

```
<?php
$titles = $books->xpath('book/title');
foreach($titles as $title)
{
        echo $title.PHP_EOL;
}
?>
```

All ISBN numbers would be retrieved like this:

```
<?php
$isbn = $books->xpath('book/@isbn');
foreach($isbn as $isbn)
{
        echo $isbn.PHP_EOL;
}
?>
```

You can of course use any valid XPath query. The method will always return an array.

Parsing RSS feeds

Now let's try to parse some real-world data. We'll use the RSS feed from the forums.

```
<?php
$rss = new
SimpleXMLElement('http://www.phpfreaks.com/forums/index.php?type=rss;action=.
xml', null, true);

echo "<h1><a href='{$rss->channel->link}'>{$rss->channel-
>title}</a></h1>".PHP_EOL.'<hr />'.PHP_EOL;

foreach($rss->xpath('channel/item') as $item)
{
        echo <<<EOF
<h2><a href='{$item->link}'>{$item->title}</a></h2>
<div>Posted at: {$item->pubDate}</div>
{$item->description}
<hr />

EOF;
}
?>
```

This was in fact the RSS reader I promised you on page 1. If you expected something fancy I suppose I've disappointed you. If you want to make it fancy then just get started, add some AJAX, a spiffy interface and soon you'll be competing with Google Reader.

Parsing data of unknown structure

As you see, parsing XML data is simple with `SimpleXML` when you already know the structure of the XML data, but what if you want to parse something which you don't already know the structure of? We'll take a look at that now, but we'll still use our `books.xml` file.

```
<?php
function parse_recursive(SimpleXMLElement $element, $level = 0)
{
        // determine how much we'll indent
        $indent     = str_repeat("\t", $level);

        // get the value and trim any whitespace from the start and end
        $value      = trim((string) $element);
        $attributes = $element->attributes();   // get all attributes
        $children   = $element->children();     // get all children

        echo "{$indent}Parsing '{$element->getName()}'...".PHP_EOL;
        if(count($children) == 0 && !empty($value))
        // only show value if there is any and if there aren't any children
        {
                echo "{$indent}Value: {$element}".PHP_EOL;
        }

        // only show attributes if there are any
        if(count($attributes) > 0)
        {
                echo $indent.'Has '.count($attributes).'
                     attribute(s):'.PHP_EOL;
                foreach($attributes as $attribute)
                {
                     echo "{$indent}-
                          {$attribute->getName()}:{$attribute}".PHP_EOL;
                }
        }

        // only show children if there are any
        if(count($children))
        {
                echo $indent.'Has '.count($children).' child(ren):'.PHP_EOL;
                foreach($children as $child)
                {
                        parse_recursive($child, $level+1); // recursion :)
                }
        }

        echo $indent.PHP_EOL; // just to make it "cleaner"
}

$xml = new SimpleXMLElement('books.xml', null, true);

parse_recursive($xml);
?>
```

Note: If you wish run that through your browser make sure to add `header('Content-type: text/plain');` before anything is output so the browser will know it's plain text and not HTML we are serving.

We've used recursion which means that we theoretically will be able to parse any XML document no matter how many times nested data there is (this is of course not possible due to script execution time etc.).

In the script I decided that if there are any children then we won't display any possible value. This is because of the way I've decided to output it. Had you chosen to use HTML to output it or chosen to put it in an array then you could have taken the value as well.

That's it for parsing XML documents. Next we'll change documents.

Altering XML data with PHP DOM

Another thing I would like to introduce you to is PHP DOM. It's much more powerful than SimpleXML and we'll be using this for the remainder of the tutorial.

Adding nodes

Still using `books.xml` we now want to add a new how many pages there are in each book and save it as `books2.xml`.

```
<?php
// isbn => pages
$page_numbers = array(
                '978-1594489501' => '384', // A Thousand Splendid Suns
                '978-1594489587' => '352', // The Brief Wondrous Life of
Oscar Wao
                '978-0545010221' => '784', // Harry Potter and the Deathly
Hallows
        );

$dom = new DOMDocument();
$dom->load('books.xml');
$xpath = new DOMXPath($dom);

$books = $xpath->query('book');
foreach($books as $book)
{
        $book->appendChild($dom->createElement('pages',
                        $page_numbers[$book->getAttribute('isbn')]));
}

$dom->save('books2.xml');
?>
```

First we have an array of ISBN numbers and page numbers. Then we create a new instance of `DOMDocument` and load our file into it. Next we'll create an instance of DOMXPath and pass our `$dom` object to it. Then we run an XPath query saying "select all nodes elements called 'book'".

This will return an instance of `DOMNodelist`. If you want to access a specific item then you can use `DOMNodelist::item()`. That method takes one argument, the index of the element you wish to retrieve. The indices are named similarly to the indices of an array (starting from 0).

Inside the loop we do the following:

1. Get the ISBN number (an attribute)
2. Get the number of pages using the ISBN number from our array
3. Create a new `DOMElement` with the value we just retrieved
4. Append the new element as a child to `$book`

Finally we save the file as `books2.xml`. We could have chosen to output it instead, and then the last line should have been replaced with:

```
echo $dom->saveXML();
```

When selecting all the book elements we could also have used `DOMDocument::getElementsByTagName()`:

```
$books = $dom->getElementsByTagName('book');
```

Changing data

Now we'd like to change the author names so it's in a "Last name, First name" format instead of "First name Last name". In our small file it would be easy to just change it, but suppose we had an XML file with thousands of entries, it would be tedious having to change it all manually. Luckily we can again use PHP for this purpose as we will now be looking at how you change data. The new data will be stored in `books3.xml`.

```
<?php
$dom = new DOMDocument();
$dom->load('books.xml');
$xpath = new DOMXPath($dom);

$authors = $xpath->query('book/author');
foreach($authors as $author)
{
        $a          = $author->nodeValue; // shortcut
        $last_name  = substr($a, strrpos($a, ' ')+1);
        $first_name = substr($a, 0, strrpos($a, ' '));
        $author->nodeValue = "{$last_name}, {$first_name}";
}

$dom->save('books3.xml');
?>
```

We start the same way as before. The XPath query this time says "get all elements called 'author' which are children of an element called 'book'".

The value of a node can be accessed through its property `nodeValue`. We do some simple switching around and change the node value for each author element. Finally we save again.

Deleting data

This time we want to make elements called `author_firstname` and `author_lastname` and then delete `author`. We'll save it in `books4.xml`.

```
<?php
$dom = new DOMDocument();
$dom->load('books.xml');
$xpath = new DOMXPath($dom);

$books = $xpath->query('book');
foreach($books as $book)
{
        $author     = $book->getElementsByTagName('author')->item(0);
        $a          = $author->nodeValue; // shortcut
        $last_name  = substr($a, strrpos($a, ' ')+1);
        $first_name = substr($a, 0, strrpos($a, ' '));

        $book->removeChild($author);
        $book->appendChild($dom->createElement('author_firstname',
                           $first_name));
        $book->appendChild($dom->createElement('author_lastname',
                           $last_name));
}

$dom->save('books4.xml');
?>
```

I believe the code is pretty self-explanatory. `DOMElement::removeChild()` removes a child and `DOMElement::appendChild()` appends a child. Note that you need an instance of an element in order to remove it from another element.

There is another method called `DOMElement::replaceChild()` which takes two elements: the new and the old element.

Parsing data with DOM

You can use DOM to parse data as well, but it is easier to use SimpleXML. I'll show you anyways though. We'll use `books4.xml` to create a table similar to the one we created using SimpleXML with `books.xml`.

```
<?php
$dom = new DOMDocument();
$dom->load('books4.xml');
$xpath = new DOMXPath($dom);

$books = $xpath->query('book');
echo <<<EOF
<h1>Book listing</h1>
<table>
        <tr>
                <th>Title</th>
```

```
                <th>Author</th>
                <th>Publisher</th>
                <th>Price at Amazon.com</th>
                <th>ISBN</th>
        </tr>

EOF;
foreach($books as $book)
{
        echo <<<EOF
        <tr>
                <td>{$book->getElementsByTagName('title')->item(0)-
>nodeValue}</td>
                <td>{$book->getElementsByTagName('author_firstname')-
>item(0)->nodeValue} {$book->getElementsByTagName('author_lastname')-
>item(0)->nodeValue}</td>
                <td>{$book->getElementsByTagName('publisher')->item(0)-
>nodeValue}</td>
                <td>\${$book->getElementsByTagName('amazon_price')->item(0)-
>nodeValue}</td>
                <td>{$book->getAttribute('isbn')}</td>
        </tr>

EOF;
}
echo '</table>';
?>
```

As you see, SimpleXML is easier to use for parsing XML data. Where we could get the title like this in SimpleXML:

```
$book->title;
```

We have to use this with DOM:

```
$book->getElementsByTagName('title')->item(0)->nodeValue;
```