XML for PHP developers, Part 1: The 15-minute PHP-with-XML starter

Learn how PHP5 has vastly improved how you work with XML in PHP

Cliff Morgan

March 07, 2007
(First published February 27, 2007)

This first article of a three-part series introduces PHP5's XML implementation and helps those relatively new to using XML with PHP to read, parse, and manipulate, and write a short and uncomplicated XML file using the DOM and SimpleXML in a PHP environment.

View more content in this series

Introduction

Other articles in this series
- XML for PHP developers, Part 2: Advanced XML parsing techniques
- XML for PHP developers, Part 3: Advanced techniques to read, manipulate, and write XML

It's hard to miss the importance of XML in today's application development environment. If you've never before worked with XML in PHP or have not yet made the jump to PHP5, this starter guide to working with new functionality available in PHP5 for XML might persuade you just how easy to work with XML can be. This first article in a three-part series, focusing on quick start API's, demonstrates how SimpleXML, in combination where necessary with the DOM, is the ideal choice for developers working with straightforward, predictable, and relatively small XML documents. These documents are exactly the sort passed by Ajax applications containing, for example, the contents of a form submission or perhaps the response of a Web service application programming interface (API) like weather.com.

XML fundamentals

Some background about XML will allow you to understand its importance to the PHP developer and allow you to understand and create straightforward XML documents.
About XML
Extensible Markup Language (XML) is described as both a markup language and a text based data storage format, depending on who you talk to. It is a subset of Standard Generalized Markup Language (SGML); it offers a text-based means to apply and describe a tree-based structure to information. XML serves as the basis for a number of languages/formats, such as Really Simple Syndication (RSS), Mozilla's XML User Interface Language (XUL), Macromedia's Maximum eXperience Markup Language (MXML), Microsoft's eXtensible Application Markup Language (XAML), and the open source Java XML UI Markup Language (XAMJ). As the many flavors of XML demonstrate, XML is a big deal. Everyone wants to get on the XML bandwagon.

Writing XML
XML's basic unit of data is the element. Elements are delimited by a start tag, such as <book>, and an end tag, such as </book>. If you have a start tag, you must have an end tag. If you fail to include an end tag for each start tag, your XML document is not well-formed, and parsers will not parse the document properly. Tags are usually named to reflect the type of content contained in the element. You would expect an element named book to contain a book title, such as Great American Novel (see Listing 1). The content between the tags, including the white spaces, is referred to as character data.

Listing 1. A sample XML document

```xml
<books>
  <book>
    <title>Great American Novel</title>
    <characters>
      <character>
        <name>Cliff</name>
        <desc>really great guy</desc>
      </character>
      <character>
        <name>Lovely Woman</name>
        <desc>matchless beauty</desc>
      </character>
      <character>
        <name>Loyal Dog</name>
        <desc>sleepy</desc>
      </character>
    </characters>
    <plot>
      Cliff meets Lovely Woman. Loyal Dog sleeps, but wakes up to bark at mailman.
    </plot>
    <success type="bestseller">4</success>
    <success type="bookclubs">9</success>
  </book>
</books>
```

XML element and attribute names can consist of the upper case alphabet A-Z, the lower case alphabet a-z, digits 0-9, certain special and non-English characters, and three punctuation marks, the hyphen, the underscore, and the period. Other punctuation marks are not allowed in names.

XML is case sensitive. In this example, <Book> and <book> describe two different elements. Either is an acceptable element name. It's probably not a good idea to use <Book> and <book> to describe two different elements, as the possibility of clerical error seems high.
Each XML document contains one and only one root element. The root element is the only element in an XML document that does not have a parent. In the example above, the root element is <books>. Most XML documents contain parent and child elements. The <books> element has one child, <book>. The <book> element has four children, <title>, <characters>, <plot>, and <success>. The <characters> element has three child elements, each of which is a <character> element. Each <character> element has two child elements, <name> and <desc>.

In addition to the nesting of elements that create the parent-child relationships, XML elements can also have attributes. Attributes are name-value pairs attached to an element’s start tag. Names are separated from values by an equal sign, =. Values are enclosed by single or double quotation marks. In Listing 1 above, the <success> element possesses two attributes, "bestseller" and "bookclubs". There are different schools of thought among XML developers about the use of attributes. Most information contained in an attribute could be contained in a child element. Some developers insist that attribute information should be metadata, namely information about the data, and not the data itself. The data itself should be contained in elements. The choice of whether to use attributes or not really depends on the nature of the data and how data will be extracted from the XML.

**Strengths of XML**

One of XML’s good qualities is its relative simplicity. You can write XML with basic text editors and word processors, no special tools or software required. The basic syntax for XML consists of nested elements, some of which have attributes and content. An element usually consists of two tags, a start tag and an end tag, each of which is bracketed by an open <tag> and a close </tag>. XML is case sensitive and does not ignore white space. It looks a lot like HTML, which is familiar to a lot of people, but, unlike HTML, it allows you to name your tags to best describe your data. Some of XML’s advantages are its self-documenting, human, and machine-readable format, its support for Unicode, which allows for internationalization in human language support, and its stringent syntax and parsing requirements. Unfortunately, UTF-8 is problematic in PHP5; this shortcoming is one of the forces driving the development of PHP6.

**Weaknesses of XML**

XML is wordy and redundant, with the attendant consequences of being large to store and a huge consumer of bandwidth. People are supposed to be able to read it, but it’s hard to imagine a human trying to read an XML file with 7 million nodes. The most basic parser functionality doesn’t support a wide array of data types; therefore, irregular or unusual data, which is common, is a primary source of difficulty.

**Well-Formed XML**

An XML document is well-formed if it follows all of XML’s syntax rules. If a document is not well-formed, it is not XML, in a technical sense. An HTML tag such as <br> is unacceptable in XML; the tag should be written <br /> to be well-formed XML. A parser won’t parse XML properly if it is not well-formed. Additionally, an XML document must have one and only one root element. Think of the one root element as being like an endless file cabinet. You have one file cabinet, but there are
few limits as to what and how much you can fit into the file cabinet. There are endless drawers and folders into which you can stuff information.

**PHP fundamentals**

Most readers of this article are already working with PHP, but might not be aware of its history and development.

**About PHP**

Hypertext Preprocessor (PHP) is cross-platform scripting language used to compose dynamic Web pages and server-side application software. It began as Personal Home Page/Form Interpreter (PHP/FI), and took on new life in the hands of Suraski and Gutmans, who launched PHP3 in June 1998. Their company, Zend Technologies, still manages the development of PHP.

PHP5 was released in July, 2004, powered by the Zend Engine II and includes many new features such as:

- New support for object-oriented programming
- Better support for MySQL
- Better support for XML, which is what you're interested in

**PHP5 and XML**

While PHP has offered XML support since its early versions, that support improved exponentially with the introduction of PHP5. Because the PHP4 support for XML was somewhat limited, such as offering only a SAX-based parser enabled by default and the PHP4 DOM not implementing the W3C standard, PHP XML developers reinvented the wheel, so to speak, with PHP5 and complied with commonly used standards.

**New for XML in PHP5**

PHP5 includes totally rewritten and new extensions, including the SAX parser, the DOM, SimpleXML, XMLReader, XMLWriter, and the XSLT processor. All these extensions are now based on the libxml2.

Along with the SAX support improved from PHP4, PHP5 also supports both the DOM according to W3C standard and the SimpleXML extension. SAX, DOM, and SimpleXML are all enabled by default. If you are familiar with the DOM from other languages, you will have an easier time coding with similar functionality in PHP than before.

**Reading, manipulating, and writing XML in PHP5**

SimpleXML, in combination where necessary with the DOM, is the ideal choice for developers working with straightforward, predictable, and relatively small XML documents to read, manipulate, and write XML in PHP5.
Quick start APIs of choice

Of the many APIs available in PHP5, the DOM and SimpleXML are the most familiar, in the case of the DOM, and the easiest to code, in the case of SimpleXML. And for the most common situations, like those you are dealing with here, the most functional.

DOM extension

The Document Object Model (DOM) is a W3C standard set of objects for representing HTML and XML documents, a standard model of how you can combine these objects, and a standard interface for accessing and manipulating them. Many vendors support the DOM as an interface to their proprietary data structures and APIs, which gives the DOM model a lot of authority with developers due to its familiarity. The DOM is easy to understand and utilize since its structure in memory resembles the original XML document. To pass on information to the application, DOM creates a tree of objects that duplicates exactly the tree of elements from the XML file, with every XML element being a node in the tree. The DOM is a tree-based parser. Because DOM builds a tree of the entire document, it uses a lot of memory and processor time. Therefore, performance issues make it impractical to parse large documents with DOM. The key use of the DOM extension in the context of this article is its ability to import SimpleXML format and output DOM format XML, or the reverse, for use as a string or XML file.

SimpleXML

The SimpleXML extension is the tool of choice for parsing an XML document. The SimpleXML extension requires PHP5 and includes interoperability with the DOM for writing XML files and built-in XPath support. SimpleXML works best with uncomplicated, record-like data, such as XML passed as a document or string from another internal part of the same application. Provided that the XML document isn't too complicated, too deep, and lacks mixed content, SimpleXML is easier to code than the DOM, as its name implies. It is also more reliable if you work with a known document structure.

Quick start examples

These are your quick start examples of working with the DOM and SimpleXML for small, vanilla XML files.

The DOM in action

The DOM is the W3C DOM specification that you work with in a browser and manipulate with JavaScript. It has all the same methods, so you will use familiar coding techniques. Listing 2 illustrates the use of the DOM to create an XML string and XML document, formatted for your viewing pleasure.

Listing 2. Using the DOM

```php
<?php

//Creates XML string and XML document using the DOM

```
$dom = new 
DOMDocument('1.0');

//add root - <books>
$books = $dom->appendChild($dom->createElement('books'));

//add <book> element to <books>
$book = $books->appendChild($dom->createElement('book'));

//add <title> element to <book>
$title = $book->appendChild($dom->createElement('title'));

//add <title> text node element to <title>
$title->appendChild($dom->createTextNode('Great American Novel'));

//generate xml $dom->formatOutput = true; // set the formatOutput attribute of $domDocument to true
// save XML as string or file
$test1 = $dom->saveXML(); // put string in test1
$dom->save('test1.xml'); // save as file
?>

This produces the output file in Listing 3.

Listing 3. The output file

```xml
<?xml version="1.0"?>
<books>
  <book>
    <title>Great American Novel</title>
  </book>
</books>
```

Listing 4 imports a SimpleXMLElement object into a DOMElement object, illustrating the interoperability of the DOM and SimpleXML.

Listing 4. Interoperability, Part 1 -- DOM imports SimpleXML

```php
<?php
$sxe = simplexml_load_string('<books><book><title>'.
  'Great American Novel</title></book></books>');</n
if ($sxe === false) {
  echo 'Error while parsing the document';
  exit;
}

$dom_sxe = dom_import_simplexml($sxe);
if (!$dom_sxe) {
  echo 'Error while converting XML';
  exit;
}

$dom = new DOMDocument('1.0');
$dom_sxe = $dom->importNode($dom_sxe, true);
$dom_sxe = $dom->appendChild($dom_sxe);

echo $dom->save('test2.xml');
?>
```
The function in Listing 5 takes a node of a DOM document and makes it into a SimpleXML node. You can then use this new object as a native SimpleXML element. If any errors occur, it returns FALSE.

Listing 5. Interoperability, Part 2 -- SimpleXML imports DOM

```php
<?php
$dom = new domDocument;
if (!$dom) {
    echo 'Error while parsing the document';
    exit;
}
$s = simplexml_import_dom($dom);

echo $s->book[0]->title; // Great American Novel
?>
```

SimpleXML in action

The SimpleXML extension is the tool of choice for parsing an XML document. The SimpleXML extension includes interoperability with the DOM for writing XML files and built-in XPath support. SimpleXML is easier to code than the DOM, as its name implies.

For those of you who might be new to PHP, Listing 6 formats a test XML file as an include for your convenience.

Listing 6. Test XML file formatted as a PHP include called example.php in the following code samples

```php
<?php
$xmlstr = <<<XML
<books>
  <book>
    <title>Great American Novel</title>
    <characters>
      <character>
        <name>Cliff</name>
        <desc>really great guy</desc>
      </character>
      <character>
        <name>Lovely Woman</name>
        <desc>matchless beauty</desc>
      </character>
      <character>
        <name>Loyal Dog</name>
        <desc>sleepy</desc>
      </character>
    </characters>
    <plot>
      Cliff meets Lovely Woman. Loyal Dog sleeps, but wakes up to bark at mailman.
    </plot>
    <success type="bestseller">4</success>
    <success type="bookclubs">9</success>
  </book>
</books>
XML
```
In an Ajax application, you might want to extract the zip code from an XML document and query a database. Listing 7 extracts <plot> from your example XML include directly above.

**Listing 7. Extracting the Node -- How easy does it get?**

```php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);
echo $xml->book[0]->plot; // "Cliff meets Lovely Woman. ..."
```

On the other hand, you might want to extract a multi-line address. When multiple instances of an element exist as children of a single parent element, normal iteration techniques apply. Listing 8 demonstrates this functionality.

**Listing 8. Extracting multiple instances of an element**

```php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);
/* For each <book> node, echo a separate <plot>. */
foreach ($xml->book as $book) {
    echo $book->plot, '<br />';
}
```

In addition to reading element names and their values, SimpleXML can also access element attributes. In Listing 9, access attributes of an element just as you would elements of an array.

**Listing 9. Demonstrating SimpleXML accessing the attributes of an element**

```php
//Input XML file repeated for your convenience
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<books>
    <book>
        <title>Great American Novel</title>
        <characters>
            <character>
                <name>Cliff</name>
                <desc>really great guy</desc>
            </character>
            <character>
                <name>Lovely Woman</name>
                <desc>matchless beauty</desc>
            </character>
        </characters>
    </book>
</books>
```

<character>
  <character>
    <name>Loyal Dog</name>
    <desc)sleepy</desc>
  </character>
</characters>

<plot>
  Cliff meets Lovely Woman. Loyal Dog sleeps, but wakes up to bark at mailman.
</plot>

<?php
include 'example.php';

$xml = new SimpleXMLElement($xmlstr);

/* Access the <success> nodes of the first book. */
foreach ($xml->book[0]->success as $success) {
    switch((string) $success['type']) {
        case 'bestseller':
            echo $success, ' months on bestseller list';
            break;
        case 'bookclubs':
            echo $success, ' bookclub listings';
            break;
    }
}
?>

To compare an element or attribute with a string or pass it into a function that requires a string, you must cast it to a string using (string). Otherwise, by default, PHP treats the element as an object, as Listing 10 demonstrates.

**Listing 10. Call it a string or lose**

```php
<?php
include 'example.php';

$xml = new SimpleXMLElement($xmlstr);

if ((string) $xml->book->title == 'Great American Novel') {
    print 'My favorite book.';
}

htmlentities((string) $xml->book->title);
?>
```

Data in SimpleXML doesn't have to be constant. Listing 11 will output a new XML document, shown below, just like the original, except that the new XML will change Cliff to Big Cliff.

**Listing 11. Changing text node using SimpleXML**

```php
<?php

XML for PHP developers, Part 1: The 15-minute PHP-with-XML starter
```
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<books>
  <book>
    <title>Great American Novel</title>
    <characters>
      <character>
        <name>Cliff</name>
        <desc>really great guy</desc>
      </character>
      <character>
        <name>Lovely Woman</name>
        <desc>matchless beauty</desc>
      </character>
      <character>
        <name>Loyal Dog</name>
        <desc>sleepy</desc>
      </character>
    </characters>
    <plot>
      Cliff meets Lovely Woman. Loyal Dog sleeps, but wakes up to bark at mailman.
    </plot>
    <success type="bestseller">4</success>
    <success type="bookclubs">9</success>
  </book>
</books>
XML;

<?php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);
$xml->book[0]->characters->character[0]->name = 'Big Cliff';
echo $xml->asXML();
?>

Since PHP 5.1.3, SimpleXML has had the ability to easily add children and attributes. Listing 12 will output an XML document based on the original but having a new character and descriptor.

**Listing 12. Adding children and text nodes using SimpleXML**

<?php
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<books>
  <book>
    <title>Great American Novel</title>
    <characters>
      <character>
        <name>Cliff</name>
        <desc>really great guy</desc>
      </character>
      <character>
        <name>Lovely Woman</name>
        <desc>matchless beauty</desc>
      </character>
      <character>
        <name>Loyal Dog</name>
        <desc>sleepy</desc>
      </character>
    </characters>
    <plot>
      Cliff meets Lovely Woman. Loyal Dog sleeps, but wakes up to bark at mailman.
    </plot>
    <success type="bestseller">4</success>
    <success type="bookclubs">9</success>
  </book>
</books>
XML;

<?php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);
$xml->book[0]->characters->character[0]->name = 'Big Cliff';
echo $xml->asXML();
?>
<?php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);

$character = $xml->book[0]->characters->addChild('character');
$character->addChild('name', 'Yellow Cat');
$character->addChild('desc', 'aloof');

$success = $xml->book[0]->addChild('success', '2');
$success->addAttribute('type', 'reprints');

echo $xml->asXML();
?>

Summary

This first article in a three-part series, focusing on quick start APIs, demonstrates how SimpleXML, in combination where necessary with the DOM, is the ideal choice for developers working with straightforward, predictable, and relatively small XML documents. PHP5 has vastly improved the developer's ability to work with XML in PHP. Part 2 will focus on advanced XML parsing techniques.
Related topics

- **XML for PHP developers, Part 2: Advanced XML parsing techniques** (Cliff Morgan, developerWorks, March 2007): In Part 2 of this three-part series, explore XML parsing techniques in PHP5, and learn how to decide which parsing method is best for your app.
- **XML for PHP developers, Part 3: Advanced techniques to read, manipulate, and write XML** (Cliff Morgan, developerWorks, March 2007): Learn more techniques to read, manipulate, and write XML in PHP5 in this final article of a three-part series on XML for PHP developers.
- **Reading and writing the XML DOM in PHP** (Jack Herrington, developerWorks, December 2005): Read XML with three methods: the DOM library, the SAX parser, and regular expressions. Plus, learn to write XML using DOM and PHP text templating.
- **SimpleXML Processing with PHP** (Elliottte Rusty Harold, developerWorks, October 2006): Try the SimpleXML extension and enable your PHP pages to query, search, modify, and republish XML.
- For the first of a three-part article series on SimpleXML, read **Introducing Simple XML in PHP5** (Alejandro Gervasio, Dev Shed, June 2006): Save work with the basics of the simplexml extension in PHP 5, a library that primarily focuses on parsing simple XML files.
- **PHP Cookbook, Second Edition** (Adam Trachtenberg and David Sklar, O'Reilly Media, August 2006): Learn to build dynamic Web applications that work on any Web browser.
- **XML.com**: Visit O'Reilly's XML site for comprehensive coverage of the XML world.
- **W3C XML Information**: Read the XML specification from the source.
- **PHP development home site**: Learn more about this widely-used general-purpose scripting language that is especially suited for Web development.
- Visit **PEAR: PHP Extension and Application Repository**: Get more information on PEAR, a framework and distribution system for reusable PHP components.
- **PECL: PHP Extension Community Library**: Visit the sister site to PEAR and repository for PHP Extensions.
- **Planet PHP**: Visit the PHP developer community news source.
- **xmlib2**: Get the the XML C parser and toolkit of Gnome.
- **XML technical library**: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **IBM trial software**: Build your next development project with trial software available for download directly from developerWorks.

© Copyright IBM Corporation 2007

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)