SimpleXML processing with PHP

A markup-specific library for XML processing in PHP

Elliotte Rusty Harold

October 10, 2006

Discover the SimpleXML extension, which is bundled with PHP version 5 and enables PHP pages to query, search, modify, and republish XML in a PHP-friendly syntax.

PHP version 5 introduced SimpleXML, a new application programming interface (API) for reading and writing XML. In SimpleXML, expressions such as:

```php
$doc->rss->channel->item->title
```

select elements from a document. As long as you have a good idea of your document's structure, such expressions are easy to write. However, if you don't know exactly where the elements of interest appear (as might be the case in Docbook, HTML, and similar narrative documents), SimpleXML can use XPath expressions to find the elements.

Starting with SimpleXML

Suppose you want a PHP page that converts an RSS feed into HTML. RSS is a basic XML format for publishing syndicated content. The root element of the document is `rss`, which contains a single `channel` element. The `channel` element contains metadata about the feed, including its title, language, and URL. It also contains various stories enclosed in `item` elements. Each `item` has a `link` element containing a URL and either a `title` or a `description` (usually both) that contain plain text. Namespaces are not used. There's more to RSS than that, but this is all you need to know for this article. Listing 1 shows a typical example with a couple of news items.
### Listing 1. An RSS feed

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rss
    version="0.92">
  <channel>
    <title>Mokka mit Schlag</title>
    <link>http://www.elharo.com/blog</link>
    <language>en</language>
    <item>
      <title>Penn Station: Gone but not Forgotten</title>
      <description>The old Penn Station in New York was torn down before I was born. Looking at these pictures, that feels like a mistake. The current site is functional, but no more; really just some office towers and underground corridors of no particular interest or beauty. The new Madison Square... </description>
      <link>http://www.elharo.com/blog/new-york/2006/07/31/penn-station</link>
    </item>
    <item>
      <title>Personal for Elliotte Harold</title>
      <description>Some people use very obnoxious spam filters that require you to type some random string in your subject such as E37T to get through. Needless to say neither I nor most other people bother to communicate with these paranoids. They are grossly overreacting to the spam problem. Personally I won't ...</description>
      <link>http://www.elharo.com/blog/tech/2006/07/28/personal-for-elliotte-harold</link>
    </item>
  </channel>
</rss>
```

Let's develop a PHP page that formats any RSS feed as HTML. **Listing 2** shows the skeleton for what the page will look like.

### Listing 2. The static skeleton for the PHP code

```php
<?php // Load and parse the XML document ?>

<html xml:lang="en" lang="en">
<head>
  <title><?php // The title will be read from the RSS ?></title>
</head>

<body>
  <?php // The title will be read from the RSS again ?></body>

<?php // Here we'll put a loop to include each item's title and description ?>
</html>
```

### Parsing the XML document

The first step is to parse the XML document and store it in a variable. Doing so takes only a single line of code that passes a URL to the `simplexml_load_file()` function:

```php
$rss = simplexml_load_file('http://partners.userland.com/nytRss/nytHomepage.xml');
```

### A word of warning

The scheme used here is dangerously sub-optimal. I really shouldn't load and parse the RSS feed every time the page is hit. This is slow for readers of the page and a potential denial of service against the RSS feeds I'm loading, most of which specify a maximum refresh rate of approximately once per hour. A real solution should cache either the generated HTML page, the RSS feeds, or both. However, that issue is orthogonal to using the SimpleXML library, so I gloss over it here.

For this example, I've populated the page from Userland's *New York Times* feed at http://partners.userland.com/nytRss/nytHomepage.xml. Of course, you could use any URL for any other RSS feed instead.

Note that despite the name `simplexml_load_file()`, this function will indeed parse an XML document at a remote HTTP URL. This isn't the only surprise in this function, either. The return value -- here stored in the `$rss` variable -- does not point to the entire document, as you might expect from experience with other APIs such as the Document Object Model (DOM). Rather, it
points to the root element of the document. Content found in the prolog and epilog of the document is inaccessible from SimpleXML.

**Finding the feed title**

The title for the entire feed (as distinct from the titles of the individual stories in the feed) resides in the `title` child of the `channel` child of the `rss` root element. You can load this title as though the XML document were simply the serialized form of an object of class `rss`, with a `channel` field that itself had a `title` field. Using the regular PHP object reference syntax, this statement finds the title:

```
$title = $rss->channel->title;
```

Having found the title, you must add it to the output HTML. Doing so is easy: Simply echo the `$title` variable:

```
<title><?php echo $title; ?></title>
```

This line outputs the string-value of the element, not the entire element. That is, the text content is written but the tags are not.

You can even skip the intermediate `$title` variable completely:

```
<title><?php echo $rss->channel->title; ?></title>
```

Because this page reuses that value in several places, I find it more convenient to store it in a descriptively named variable.

**Iterating through the items**

Next, you must find the items in the feed. The expression that performs this task is obvious:

```
$rss->channel->item
```

However, feeds generally contain more than one item. There might even be none of them. Consequently, this statement returns an array, which you can iterate across with a `for-each` loop:

```
foreach ($rss->channel->item as $item) { echo "<h2>", $item->title . "</h2>"; echo "<p>", $item->description . "</p>"; }
```

You can easily add links by reading the `link` element value from the RSS feed. Just output an `a` element from the PHP, and use `$item->link` to retrieve the URL. **Listing 3** adds this element and fills in the skeleton from **Listing 1**.
Listing 3. A simple but complete PHP RSS reader

```php
<?php // Load and parse the XML document
    $rss = simplexml_load_file('http://partners.userland.com/nytRss/nytHomepage.xml');
    $title = $rss->channel->title; ?>
<html xml:lang="en" lang="en">
<head> <title><?php echo $title; ?></title> </head>
<body> <h1><?php echo $title; ?></h1> <?php // Here we'll
    put a loop to include each item's title and description foreach
    ($rss->channel->item as $item) { echo "<h2><a href="" .
        $item->link . "">" . $item->title . "</a></h2>"; echo
        "<p>" . $item->description . "</p>"; } ?></body>
</html>
```

That's all it takes to write a simple RSS reader in PHP -- a few lines of HTML and a few lines of PHP. Not counting white space, it's a total of only 20 lines. Of course, this is not the most feature-rich, optimized, or robust implementation. Let's see what we can do to fix that.

Error handling

Not all RSS feeds are as well formed as they're supposed to be. The XML specification requires processors to stop processing documents as soon as a well-formedness error is detected, and SimpleXML is a conforming XML processor. However, it doesn't give you a lot of help when it finds an error. Generally, it logs a warning in the php-errors file (but without a detailed error message), and the `simplexml-load-file()` function returns FALSE. If you aren't confident that the file you're parsing is well formed, check for this error before using the file's data, as shown in Listing 4.

Listing 4. Watching out for malformed input

```php
<?php $rss =
    simplexml_load_file('http://www.cafeaulait.org/today.rss');
if ($rss) { foreach
    ($rss->xpath('//title') as $title) { echo "<h2>" . $title . "</h2>";
} } else { echo "Oops! The input is malformed!"; } ?>
```

The `libxml_get_errors()` method will return more helpful, debugging information about what went wrong, though these are usually not details you want to show to the end-reader.

The other common error case is where the document is indeed well formed but doesn't contain exactly the elements you expect exactly where you expect them. What happens to an expression such as `$doc->rss->channel->item->title` when an item does not have a title (as is the case in at least one top-100 RSS feed)? The simplest approach is always to treat the return value as an array and loop over it. In this case, you're covered whether there are more or fewer elements than you expect. However, if you know that you only want the first element in the document -- even if there are more than one -- you can ask for it by index, starting at zero. For example, to request the first item's title, you could write:

```php
$doc->rss->channel->item[0]->title[0]
```

If there is no first item, or if the first item does not have a title, this item is treated the same as any other out-of-bounds index in a PHP array. That is, the result is null, which is converted to the empty string when you try to insert it into the output HTML.
Recognizing and rejecting unexpected formats you aren't prepared to handle is typically the province of a validating XML parser. However SimpleXML cannot validate against a Document Type Definition (DTD) or schema. It checks only for well-formedness.

### Handling namespaces

Many sites are now switching from RSS to Atom. **Listing 5** shows an example of an Atom document. In many ways, this document is similar to the RSS example. However, there's more metadata, and the root element is `feed` instead of `rss`. The `feed` element has entries instead of items. The `content` element replaces the `description` element. Most significantly, the Atom document uses a namespace, while the RSS document does not. In this way, the Atom document can embed real, un-escaped Extensible HTML (XHTML) in its content.

**Listing 5. An Atom document**

```xml
<?xml version="1.0"?> <feed
   xmlns="http://www.w3.org/2005/Atom" xml:lang="en-US"
   xml:base="http://www.cafeconleche.org/today.atom">
   <updated>2006-08-04T16:00:04-04:00</updated>
   <id>http://www.cafeconleche.org/</id> <title>Cafe con Leche XML News and Resources</title> <link rel="self" type="application/atom+xml" href="/today.atom"/> <rights>Copyright 2006 Elliotte Rusty Harold</rights> <entry> <title>Steve Palmer has posted a beta of Vienna 2.1, an open source RSS/Atom client for Mac OS X. </title> <content type="xhtml"> <div xmlns="http://www.w3.org/1999/xhtml" id="August_1_2006_25279" class="2006-08-01T07:01:19Z"> <p> Steve Palmer has posted a beta of Vienna 2.1, an open source RSS/Atom client for Mac OS X. Vienna is the first reader I've found acceptable for daily use; not great but good enough. (Of course my standards for "good enough" are pretty high.) 2.1 focuses on improving the user interface with a unified layout that lets you scroll through several articles, article filtering (e.g. read all articles since the last refresh), manual folder reordering, a new get info window, and an improved condensed layout. </p> </div> </content> <link href="/#August_1_2006_25279"/>
   </entry> <entry> <title>Matt Mullenweg has released Wordpress 2.0.4, a blog engine based on PHP and MySQL. </title> <content type="xhtml"> <div xmlns="http://www.w3.org/1999/xhtml" id="August_1_2006_21750" class="2006-08-01T06:02:30Z"> <p> Matt Mullenweg has released Wordpress 2.0.4, a blog engine based on PHP and MySQL. </p> </div> </content> <link href="/#August_1_2006_21750"/>
   </entry>
 </feed>
```

Although the element names have changed, the basic approach to handling an Atom document with SimpleXML is the same as for handling RSS. The one difference is that you must now specify a namespace Uniform Resource Identifier (URI) when requesting a named element as well as a local name. This is a two-step process: First, request the child elements in a given namespace by passing the namespace URI to the `children()` function. Then, request the elements with the right local name in that namespace. Suppose you first load the Atom feed into the variable `$feed`, like so:

```php
$feed = simplexml_load_file('http://www.cafeconleche.org/today.atom');
```
These two lines now find the title element:

```
$children = $feed->children('http://www.w3.org/2005/Atom');
$title = $children->title;
```

You can condense this code into a single statement if you like, though the line gets a bit long. All other elements in namespaces must be handled similarly. **Listing 6** shows a complete PHP page that displays the titles from a namespaced Atom feed.

**Listing 6. A simple PHP Atom headline reader**

```php
<?php
$feed = simplexml_load_file('http://www.cafeconleche.org/today.atom');
$children = $feed->children('http://www.w3.org/2005/Atom');
$title = $children->title;
?>

<html xml:lang="en" lang="en">
<head>
<title><?php echo $title; ?></title>
</head>
<body>
<h1><?php echo $title; ?></h1>
<?php
    $entries = $children->entry;
    foreach ($entries as $entry) {
        $details = $entry->children('http://www.w3.org/2005/Atom');
        echo "<h2>" . $details->title . "</h2>";
    }
?>
</body>
</html>
```

**Mixed content**

Why did I only display the headlines in this example? Because in Atom, the content of an entry can contain the full text of the story -- and not just the plain text, either, but all the markup. This is a *narrative structure*: words in a row meant for people to read. Like most such data, it has a lot of mixed content. The XML isn't so simple any more, and thus the SimpleXML approach begins to show some flaws. It can't handle mixed content in any reasonable way, and this omission rules it out for many use cases.

You can do one thing, but it's only a partial solution and works only because the `content` element contains real XHTML. You can copy that XHTML as unparsed source code straight into the output using the `asXML()` function, like so:

```php
echo "<p>
" . $details->content->asXML() . "</p>";
```

What this generates is something like **Listing 7**.

**Listing 7. Output from asXML**

```xml
<content type="xhtml">
   <div
   xmlns="http://www.w3.org/1999/xhtml" id="August_7_2006_31098" 
   class="2006-08-07T09:38:18Z">
   <p>Nikolai Grigoriev has released <a shape="rect" href="http://www.grigoriev.ru/svgmath">SVGMath 0.3</a>, a presentation MathML formatter that produces SVG written in pure Python and published under an MIT license. According to Grigoriev, "The new version can work with multiple-namespace documents (e.g. replace all MathML subtrees with SVG in an XSL-FO or XHTML document); configuration is made more flexible, and several bugs are fixed. There is also a stylesheet to adjust the vertical position of the resulting SVG image in XSL-FO."</p>
</div>
</content>
```

This isn't pure XHTML. The `content` element snuck in from the Atom document, and you'd really rather not have it. Even worse, it comes in with the wrong namespace, so it can't be recognized for what it is. Fortunately, this extra element doesn't do a great deal of practical harm, because Web
browsers simply ignore any tags they don't recognize. The finished document is invalid, but that
doesn't really matter much. If it truly bothers you, strip it out with string operations, like so:

```php
$description = $details->content->asXML(); $tags = array('<content type="xhtml">', '</content>'); $notags = array('', '');
$description = str_replace($tags, $notags, $description);
```

To make this code a bit more robust, use a regular expression rather than assuming that the start-
tag is exactly as shown above. In particular, you can account for a variety of possible attributes:

```php
// end-tag is fixed in form so it's easy to replace
$description = str_replace('</content>', '', $description); // remove start-tag, possibly
including attributes and white space
$description = ereg_replace("<content[^>]*>", '', $description);
```

Even with this improvement, your code can still trip on comments, processing instructions, and
CDATA sections. Any way you slice it, I'm afraid this is no longer so simple. Mixed content simply
exceeds the bounds of what SimpleXML was designed to handle.

**XPath**

Expressions such as `$rss->channel->item->title` are great as long as you know exactly which
elements are in the document and exactly where they are. However, you don't always know that.
For instance, in XHTML, heading elements (h1, h2, h3, and so on) can be children of the body,
a div, a table, and several other elements. Furthermore, divs, tables, blockquotes, and other
elements can nest inside each other multiple times. For many less-determinate use cases, it's
easier to use XPath expressions such as `//h1` or `//h1[contains('Ben')]`. SimpleXML enables this
functionality through the `xpath()` function.

**Listing 8** shows a PHP page that lists all the titles in an RSS document -- both the title of the feed
itself and the titles of the individual items.

**Listing 8. Using XPath to find title elements**

```html
<html xml:lang="en" lang="en">  
<head>    
<title>XPath Example</title>  
</head> 
<body>  
<?php $rss = simplexml_load_file('http://partners.userland.com/nytRss/nytHomepage.xml'); foreach ($rss->xpath('//title') as $title) { echo "<h2>" . $title . "</h2>"; } ?>  
</body> 
</html>
```

SimpleXML only supports XPath location paths and unions of location paths. It does not support
XPath expressions that do not return node-sets, such as `count(//para)` or `contains(title)`.

Starting in PHP version 5.1, SimpleXML can make XPath queries against namespaced
documents. As always in XPath, the location path must use namespace prefixes even if the
searched document uses the default namespace. The `registerXPathNamespace()` function
associates a prefix with a namespace URI for use in the next query. For example, if you wanted to
find all the title elements in an Atom document, you'd use code like that in **Listing 9**.
Listing 9. Using XPath with namespaces

```php
$atom = simplexml_load_file('http://www.cafeconleche.org/today.atom');
$atom->registerXPathNamespace('atm', 'http://www.w3.org/2005/Atom');
$titles = $atom->xpath('//atm:title');
foreach ($titles as $title) {
    echo "<h2>" . $title . "</h2>";
}
```

One final warning: XPath in PHP is quite slow. Page loads went from essentially unnoticeable to several seconds when I switched over to this XPath expression, even on an unloaded local server. If you use these techniques, you must use some sort of caching to get reasonable performance. Dynamically generating every page just won't work.

Conclusion

SimpleXML is a useful addition to the PHP programmer's toolkit provided you don't need to handle mixed content. That covers a lot of use cases. In particular, it works well with simple, record-like data. As long as the document isn't too deep, too complex, and doesn't have mixed content, SimpleXML is much easier than the DOM alternative. It also helps if you know your document structure in advance, although XPath can go a long way toward relaxing that requirement. The omission of validation and the lack of any support for mixed content is troubling but not always crippling. Many simple formats don't have mixed content, and many use cases involve only very predictable data formats. If that describes your work, you owe it to yourself to try SimpleXML. With a little attention to error handling and some effort on the caching end to alleviate performance problems, SimpleXML can be a reliable and robust means of processing XML from within PHP.
Related topics

- **Official documentation for SimpleXML:** Dig into the PHP version 5 manual for info on this toolset to convert XML to an object that you can process with normal property selectors and array iterators.
- **XML in a Nutshell** (Elliotte Rusty Harold and W. Scott Means, O'Reilly, 2005): Read about XML and XPath in depth in this concentrated intro to most things XML.
- "**An overview of the Atom version 1.0 Syndication Format**" (James Snell, developerWorks, August 2005): Look at the technical strengths of the popular Atom Syndication Format relative to other syndication formats, and examine several compelling use case examples.
- **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 2006
**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)