08 May

Introduction To SimpleXML With PHP (blog-post.html)

By Kevin Waterson 671518 12 Comments

By Kevin Waterson

Contents

Abstract

XML creation and manipulation can be quite complex and so, PHP has provided several methods of handling XML. Each method has a varying degree of complexity, but perhaps the simplest of them all, is the appropriately named extension, SimpleXML

This tutorial requires a fundamental understanding of XML structures. If you lack this, it is highly recommended brushing up on XML and how it can be used.

Getting Started

Perhaps the best place to begin, is with an XML file. For the purpose of this tutorial, an XML tree will be created which represents an address book. The file will named address.xml during this tutorial. Here is one we prepared earlier..

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<users>
  <user>
    <firstname>Sheila</firstname>
    <surname>Green</surname>
    <address>2 Good St</address>
    <city>Campbelltown</city>
    <country>Australia</country>
    <contact>
      <phone>1234 1234</phone>
      <url>http://example.com</url>
      <email>pamela@example.com</email>
    </contact>
    </user>
    <user>
    <firstname>Bruce</firstname>
    <surname>Smith</surname>
    <address>1 Yakka St</address>
    <city>Meekatharra</city>
    <country>Australia</country>
    <contact>
      <phone>4444 4444</phone>
      <url>http://yakka.example.com</url>
      <email>bruce@yakka.example.com</email>
    </contact>
  </user>
  <user>
    <firstname>Davo</firstname>
    <surname>White</surname>
    <address>The Only Way</address>
    <city>Whitefall</city>
    <country>Australia</country>
    <contact>
      <phone>8888 8888</phone>
      <url>http://white.example.com</url>
      <email>davo@white.example.com</email>
    </contact>
  </user>
  <user>
    <firstname>Shazza</firstname>
    <surname>Green</surname>
    <address>222 Great Western Hwy</address>
    <city>Bathurst</city>
    <country>Australia</country>
    <contact>
      <phone>6666 6666</phone>
      <url>http://shazza.green.example.com</url>
      <email>shazza@green.example.com</email>
    </contact>
  </user>
</users>
```

This is an example of any XML file that may be found for user data in an address book, or contact list or other data source. Note that in the content node, there are several sub nodes containing the phone, url, and email for the user. SimpleXML will allow traversing this XML structure in one simple pass as will be demonstrated here.

Load an XML file

Loading an XML file with simpleXML is achieved with a single function call to the simplexml_load_file() function. Once an XML file is loaded into SimpleXML, the data can be accessed.

```
<?php

    if( ! $xml = simplexml_load_file('address.xml') )
    {
        echo 'unable to load XML file';
    }
    else
    {
        echo 'XML file loaded successfully';
    }
?>
```

Similarly, if the XML data is in a string, rather than a file, the simplexml_load_string() function can be used in the same fashion. In this example, the XML is shortened a little for the sake of sanity, but the functionality remains the same

```
<?php
$xml_string = '<?xml version="1.0" encoding="iso-8859-1"?>
<users>
  <user>
    <firstname>Sheila</firstname>
    <surname>Green</surname>
    <address>2 Good St</address>
    <city>Campbelltown</city>
    <country>Australia</country>
    <contact>
      <phone>1234 1234</phone>
      <url>http://example.com</url>
      <email>pamela@example.com</email>
    </contact>
    </user>
</users>';

    if( ! $xml = simplexml_load_string( $xml_string ) )
    {
        echo 'Unable to load XML string';
    }
    else
    {
        echo 'XML String loaded successfully';
    }

?>
```

Regardless of which function is used to load the XML data, the resulting xml object is the same.

Accessing Nodes

; In the above script, the XML file was loaded using the simplexml_load_file() function. With the file loaded successfully, the data in each of the nodes can be accessed, and displayed, or assigned to variables, or manipulated any way program requires.

Each node is transformed into a corresponding variable name, which contains the data within that node. In the case of nested variables such as those nodes in the contact section of the XML, an easy path is provided to access these values, as if in a tree structure.

```
<?php

    if( ! $xml = simplexml_load_file('address.xml') )
    {
        echo 'unable to load XML file';
    }
    else
    {
        foreach( $xml as $user )
        {
            echo 'Firstname: '.$user->firstname.'<br />';
            echo 'Surname: '.$user->surname.'<br />';
            echo 'Address: '.$user->address.'<br />';
            echo 'City: '.$user->city.'<br />';
            echo 'Country: '.$user->country.'<br />';
            echo 'Email: '.$user->contact->phone.'<br />';
            echo 'Email: '.$user->contact->url.'<br />';
            echo 'Email: '.$user->contact->email.'<br />';
        }
    }

?>
```

Importing DOM

As seen earlier, loading XML from a file or a string is simply a matter of a single call to the simple_load_string() or the simplexml_load_file() functions. SimpleXML also provides a method to import XML which has been loaded via the DOM extension.

```
<?php
$xml_string = '<?xml version="1.0" encoding="iso-8859-1"?>
<users>
  <user>
    <firstname>Sheila</firstname>
    <surname>Green</surname>
    <address>2 Good St</address>
    <city>Campbelltown</city>
    <country>Australia</country>
    <contact>
      <phone>1234 1234</phone>
      <url>http://example.com</url>
      <email>pamela@example.com</email>
    </contact>
    </user>
</users>';

/*** a new DOM object ***/
$dom = new DOMDocument;
```

```
/*** load the XML string ***/
$dom->loadXML( $xml_string );
$sxe = simplexml_import_dom($dom);

echo $sxe->user[0]->surname;
?>
```

The script above loads the XML string via the DomDocument::loadXML method. This object can now be imported directly into SimpleXML with the <span class="codechar">simplexml_import_dom() function. The result set is an array of objects, each containing a user node. Each node within the array may accessed with the array key, in this example, the array key is zero.

Create a SimpleXMLElement

So far, this tutorial has dealt with the ways of loading XML into SimpleXML. Here an element named user is created and can then be populated with children. The initial element in this example is created with the DOM extension as SimpleXML has no method of creating the initial XML, only loading it. Of course, this initial step could be achieved with the simplexml_load_file() or simplexml_load_string functions, but here all the XML will be created pragmatically.

```
<?php

try
{
    /*** a new dom object ***/
    $dom = new domDocument;

    /*** make the output tidy ***/
    $dom->formatOutput = true;

    /*** create the root element ***/
    $root = $dom->appendChild($dom->createElement( "user" ));

    /*** create the simple xml element ***/
    $sxe = simplexml_import_dom( $dom );

    /*** add a firstname element ***/
    $sxe->addChild("firstname", "John");
    /*** add a surname element ***/
    $sxe->addChild("surname", "Brady");

    /*** echo the xml ***/
    echo $sxe->asXML();
}
catch( Exception $e )
{
    echo $e->getMessage();
}
?>
```

The above script creates a parent user element and then a firstname and surname child elements. and looks like this.

```
<?xml version="1.0"?>
<user>
  <firstname>John</firstname>
  <surname>Brady</surname>
</user>
```

Now lets populate the XML to look like the XML used earlier in this tutorial.

```
<?php

try
{
    /*** a new dom object ***/
    $dom = new domDocument;

    /*** make the output tidy ***/
    $dom->formatOutput = true;

    /*** create the root element ***/
    $root = $dom->appendChild($dom->createElement( "user" ));

    /*** create the simple xml element ***/
    $sxe = simplexml_import_dom( $dom );

    /*** add a firstname element ***/
    $sxe->addChild("firstname", "John");

    /*** add a surname element ***/
    $sxe->addChild("surname", "Brady");

    /*** add address element ***/
    $sxe->addChild("address", "1 Bunch St");

    /*** add the city element ***/
    $sxe->addChild("city", "Downtown");

    /*** add the country ***/
    $sxe->addChild("country", "America");

    /*** echo the xml ***/
    echo $sxe->asXML();
}
catch( Exception $e )
{
    echo $e->getMessage();
}
?>
```

Well, no surprises there, just the addition of a few more elements as was shown previously. The XML itself now looks like this.

```
<?xml version="1.0"?>
<user>
  <firstname>John</firstname>
  <surname>Brady</surname>
  <address>1 Bunch St</address>
  <city>Downtown</city>
  <country>America</country>
</user>
```

It is at this point, a limitation of SimpleXML is reached. If the XML above is compared to the initial XML in this tutorial, it is plain to see the contact details are missing. The limitation of is that SimpleXML cannot add nodes. To achieve this, the DOM extension needs to be used, remember, this is SimpleXML, not FullyFeaturedXML as you get when using the DOM extension. Whilst SimpleXML provides a fast and efficient method for reading XML, its abilities to create detailed XML trees is somewhat limited.

Having said that it cannot be done, here is how to do it. Rather than adding a whole node, the tree can be created by adding children to each element. Each element represents a SimpleXML object and can be used to add a child. By adding each child to a parent element, the node is gradually created.

```
<?php

try
{
    /*** a new dom object ***/
    $dom = new domDocument;

    /*** make the output tidy ***/
    $dom->formatOutput = true;

    /*** create the root element ***/
    $root = $dom->appendChild($dom->createElement( "users" ));

    /*** create the simple xml element ***/
    $sxe = simplexml_import_dom( $dom );

    /*** add a user node ***/
    $user = $sxe->addchild("user");

    /*** add a firstname element ***/
    $user->addChild("firstname", "John");

    /*** add a surname element ***/
    $user->addChild("surname", "Brady");

    /*** add address element ***/
    $user->addChild("address", "1 Bunch St");

    /*** add the city element ***/
    $user->addChild("city", "Downtown");

    /*** add the country ***/
    $user->addChild("country", "America");
    $contact = $user->addChild("contact");
    $contact->addChild("phone", "4444 4444");
    $contact->addChild("url", "http://phpro.org");
    $contact->addChild("email", "brady@bunch.example.com");

    echo $sxe->asXML();
}
catch( Exception $e )
{
    echo $e->getMessage();
}
?>
```

The XML produced from the above script, now resembles that of the desired result in the beginning of this tutorial.

```
<?xml version="1.0"?>
<users>
  <user>
    <firstname>John</firstname>
    <surname>Brady</surname>
    <address>1 Bunch St</address>
    <city>Downtown</city>
    <country>America</country>
    <contact>
      <phone>4444 4444</phone>
      <url>http://phpro.org</url>
      <email>brady@bunch.example.com</email>
    </contact>
  </user>
</users>
```

Adding a second user node, now is just a matter of repeating the above steps.

```
<?php

try
{
    /*** a new dom object ***/
    $dom = new domDocument;

    /*** make the output tidy ***/
    $dom->formatOutput = true;
```

```
/*** create the root element ***/
$root = $dom->appendChild($dom->createElement( "users" ));

/*** create the simple xml element ***/
$sxe = simplexml_import_dom( $dom );

/*** add a user node ***/
$user = $sxe->addchild("user");

/*** add a firstname element ***/
$user->addChild("firstname", "John");

/*** add a surname element ***/
$user->addChild("surname", "Brady");

/*** add address element ***/
$user->addChild("address", "1 Bunch St");

/*** add the city element ***/
$user->addChild("city", "Downtown");

/*** add the country ***/
$user->addChild("country", "America");

/*** add the contact element ***/
$contact = $user->addChild("contact");

/*** add children to the contact element ***/
$contact->addChild("phone", "4444 4444");
$contact->addChild("url", "http://phpro.org");
$contact->addChild("email", "brady@bunch.example.com");


/*** add a second user node ***/
$user = $sxe->addchild("user");

/*** add a firstname element ***/
$user->addChild("firstname", "Jenna");

/*** add a surname element ***/
```

```
    $user->addChild("surname", "Taylor");

    /*** add address element ***/
    $user->addChild("address", "The Wrong Way");

    /*** add the city element ***/
    $user->addChild("city", "Sydney");

    /*** add the country ***/
    $user->addChild("country", "Australia");

    /*** add the contact element ***/
    $contact = $user->addChild("contact");

    /*** add children to the contact element ***/
    $contact->addChild("phone", "1234 1234");
    $contact->addChild("url", "http://phpro.org");
    $contact->addChild("email", "jenna@taylor.example.com");



    /*** show the xml ***/
    echo $sxe->asXML();
}
catch( Exception $e )
{
    echo $e->getMessage();
}
?>
```

The process of adding the second user node follows exactly the same as the first user node, The DOM object creates the original XML to work with, and SimpleXML takes care of the rest. The XML itself now looks like this.

```
<?xml version="1.0"?>
<users>
  <user>
    <firstname>John</firstname>
    <surname>Brady</surname>
    <address>1 Bunch St</address>
    <city>Downtown</city>
    <country>America</country>
    <contact>
      <phone>4444 4444</phone>
      <url>http://phpro.org</url>
      <email>brady@bunch.example.com</email>
      </contact>
  </user>
  <user>
    <firstname>Jenna</firstname>
    <surname>Taylor</surname>
    <address>The Wrong Way</address>
    <city>Sydney</city>
    <country>Australia</country>
    <contact>
      <phone>1234 1234</phone>
      <url>http://phpro.org</url>
      <email>jenna@taylor.example.com</email>
    </contact>
  </user>
</users>
```

Adding Attributes

In the XML created so far, the phone element has simply been a number. But, it does not tell us what type of phone it is. Is it a land line? Is it a cell/mobile or fax or satellite phone?

An attribute named type can be added as an attribute for the phone element. In SimpleXML, attributes are added with the addAttribute() method. In this following example, an attribute named type is added with a value of mobile.

```
<?php

try
{
    /*** a new dom object ***/
    $dom = new domDocument;
```

```
/*** make the output tidy ***/
$dom->formatOutput = true;

/*** create the root element ***/
$root = $dom->appendChild($dom->createElement( "users" ));

/*** create the simple xml element ***/
$sxe = simplexml_import_dom( $dom );

/*** add a user node ***/
$user = $sxe->addchild("user");

/*** add a firstname element ***/
$user->addChild("firstname", "John");

/*** add a surname element ***/
$user->addChild("surname", "Brady");

/*** add address element ***/
$user->addChild("address", "1 Bunch St");

/*** add the city element ***/
$user->addChild("city", "Downtown");

/*** add the country ***/
$user->addChild("country", "America");

/*** add the contact element ***/
$contact = $user->addChild("contact");

/*** add children to the contact element ***/
$phone = $contact->addChild("phone", "4444 4444");

/*** add an attribute to the phone element ***/
$phone->addAttribute("type", "mobile");

/*** more children for the contact element ***/
$contact->addChild("url", "http://phpro.org");
$contact->addChild("email", "brady@bunch.example.com");
```

```
    /*** show the xml ***/
    echo $sxe->asXML();
}
catch( Exception $e )
{
    echo $e->getMessage();
}
?>
```

The resulting XML, with the new type attribute for the phone element looks like this:

```
<?xml version="1.0"?>
<users>
  <user>
    <firstname>John</firstname>
    <surname>Brady</surname>
    <address>1 Bunch St</address>
    <city>Downtown</city>
    <country>America</country>
    <contact>
      <phone type="mobile">4444 4444</phone>
      <url>http://phpro.org</url>
      <email>brady@bunch.example.com</email>
    </contact>
  </user>
</users>
```

Accessing Elements

Up to this point, the focus has been on creating XML. At this point, the focus will be on various methods of accessing XML using SimpleXML. This is what SimpleXML is really good at, and makes access to nodes, elements, attributes and namespaces quite simple, as the name would suggest.

In this section, the example.xml file will be altered a little to reflect the addition of the type attribute in the phone element. The revised example.xml file now looks like this:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<users>
  <user>
    <firstname>Sheila</firstname>
    <surname>Green</surname>
    <address>2 Good St</address>
    <city>Campbelltown</city>
    <country>Australia</country>
    <contact>
      <phone type="mobile">1234 1234</phone>
      <url>http://example.com</url>
      <email>pamela@example.com</email>
    </contact>
    </user>
    <user>
    <firstname>Bruce</firstname>
    <surname>Smith</surname>
    <address>1 Yakka St</address>
    <city>Meekatharra</city>
    <country>Australia</country>
    <contact>
      <phone type="landline">4444 4444</phone>
      <url>http://yakka.example.com</url>
      <email>bruce@yakka.example.com</email>
    </contact>
  </user>
  <user>
    <firstname>Davo</firstname>
    <surname>White</surname>
    <address>The Only Way</address>
    <city>Whitefall</city>
    <country>Australia</country>
    <contact>
      <phone type="mobile">8888 8888</phone>
      <url>http://white.example.com</url>
      <email>davo@white.example.com</email>
    </contact>
  </user>
  <user>
    <firstname>Shazza</firstname>
    <surname>Green</surname>
    <address>222 Great Western Hwy</address>
    <city>Bathurst</city>
    <country>Australia</country>
    <contact>
      <phone type="fax">6666 6666</phone>
      <url>http://shazza.green.example.com</url>
      <email>shazza@green.example.com</email>
    </contact>
  </user>
</users>
```

Earlier a small example was given on accessing elements. Here this will be expanded upon to show how complex XML structures can be tamed and access to given elements simplified. After all, this is SIMPLE XML. Lets first look at how one of the elements looks inside the XML object

```
<?php

    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
        echo "Unable to load XML file";
    }
    else
    {
        /*** loop over the elements ***/
        foreach( $xml as $element )
        {
            print_r( $element );
        }

    }
?>
```

The snippet above will print out each of the SimpleXML nodes. Here is a look at just one of them.

```
SimpleXMLElement Object
(
    [firstname] => Shazza
    [surname] => Green
    [address] => 222 Great Western Hwy
    [city] => Bathurst
    [country] => Australia
    [contact] => SimpleXMLElement Object
        (
            [phone] => 6666 6666
            [url] => http://shazza.green.example.com
            [email] => shazza@green.example.com
        )
)
```

From this internal snapshot, it is easy to see how each of the elements can be accessed. For example, to access the firstname, surname, and phone number of the second user would look like this.

```
<?php

    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file('address.xml') )
    {
        echo "Unable to load XML file";
    }
    else
    {
        echo $xml->user[1]->firstname.' '.$xml->user[1]->surname.'<br />';
        echo $xml->user[1]->contact->phone;
    }
?>
```

This provides easy access to each of the individual elements in the tree, as each node is held in an array of nodes which represent the XML tree. However, a better method is available. Note that in the data structure, the phone element shows the number, but not the type attribute of the phone element. To show attributes, a little more is required..

Accessing Attributes

Attributes can be accessed in much the same way as seen with accessing elements in the previous section. Once again, an appropriately named function is provided by SimpleXML. The function attributes() makes this step easy. In the address.xml file, only the phone element has an attribute, so we can direct our query directly at that element to retrieve an attributes it has.

```
<?php

    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
    echo "Unable to load XML file";
    }
    else
    {
    $i = 0;
    /*** loop over the elements ***/
    foreach($xml as $node)
    {
        echo $xml->user[$i]->contact->phone->attributes().'<br />';
        $i++;
    }


    }
?>
```
The above query will fetch all the attributes of the phone element and show their values.

```
mobile
landline
mobile
fax
```

XPath Queries

XPath provides a standardized method to query XML regardless of the language being used to program with. To utilize xpath in SimpleXML, a single function is all that is required, amazingly named xpath(). Lets begin with a look at how xpath can be used to get the firstnames.

```
<?php
    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
        echo "Unable to load XML file";
    }
    else
    {
        /*** show the firstname element from all nodes ***/
        print_r($xml->xpath("/users/user/firstname"));
    }
?>
```
The resulting array shows an array of simpleXML objects, each containing a single element, the firstname.

```
Array
(
    [0] => SimpleXMLElement Object
        (
            [0] => Sheila
        )

    [1] => SimpleXMLElement Object
        (
            [0] => Bruce
        )

    [2] => SimpleXMLElement Object
        (
            [0] => Davo
        )

    [3] => SimpleXMLElement Object
        (
            [0] => Shazza
        )
)
```

XPath Search by Tagname

Searching a path with XPath breaks down the path into its component elements and extracts the values. A similar search to the previous could therefore be expressed like this.

```
<?php
    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
        echo "Unable to load XML file";
    }
    else
    {
        /*** show the firstname element from all nodes ***/
        print_r($xml->xpath("//firstname"));
    }
?>
```

The resulting array if objects is identical to the previous method of access the firstname elements. This functionality can be further extended by searching for a node by an elements value. For example, if all the information about Sheila was required, the tag name and value can be supplied, and SimpleXML will traverse the XML tree to retrieve the node data it belongs to.

```
<?php
    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
        echo "Unable to load XML file";
    }
    else
    {
        /*** show the firstname element from all nodes ***/
        print_r($xml->xpath("//*[firstname='Sheila']"));
    }
?>
```

The resulting array contains all the information about Sheila.

```
Array
(
    [0] => SimpleXMLElement Object
        (
            [firstname] => Sheila
            [surname] => Green
            [address] => 2 Good St
            [city] => Campbelltown
            [country] => Australia
            [contact] => SimpleXMLElement Object
                (
                    [phone] => 1234 1234
                    [url] => http://example.com
                    [email] => pamela@example.com
                )
        )
)
```

With this information, it is now just a short step to retrieve the value of just the email element, based on the firstname elements value.

```
<?php
    error_reporting(E_ALL);

    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
        echo "Unable to load XML file";
    }
    else
    {
        /*** show the firstname element from all nodes ***/
        $info = $xml->xpath("//*[firstname='Sheila']");

        /*** fetch the email address ***/
        echo $info[0]->contact->email;
    }
?>
```

This results simply in the email address being returned.

```
pamela@example.com
```

Saving the XML

All this creating, manipulating and reading of XML is all well and good. The ability to create an array of objects from an xpath query is quite clever, however, is not truly in a machine readable format. Their needs to be a method to create an XML representation of the object. To this end, the asXML() method is provided for just this purpose. In this example, the XPath query for Sheila is saved as XML, rather than the array of objects and values.

```
<?php
    /*** create a SimpleXML object ***/
    if( ! $xml = simplexml_load_file("address.xml") )
    {
        echo "Unable to load XML file";
    }
    else
    {
        /*** show the firstname element from all nodes ***/
        $info = $xml->xpath("//*[firstname='Sheila']");

        /*** initialize the string ***/
        $xml_string = '';
        /*** loop over the results ***/
        while(list( , $node) = each($info))
        {
            $xml_string .= $node->asXML(); // <c>text</c> and <c>stuff</c>
        }

        /*** output the xml ***/
        echo $xml_string;
    }
?>
```

Now, instead of the array of values and objects, the asXML() method has transformed it all into an XML representation of the data.

```
<user>
  <firstname>Sheila
  <surname>Green
  <address>2 Good St
  <city>Campbelltown
  <country>Australia
  <contact>
    <phone type="mobile">1234 1234
    <url>http://example.com
    <email>pamela@example.com
  </contact>
</user>
```