



By icarus

This article copyright [Melonfire](#) 2000–2002. All rights reserved.

Table of Contents

<u>Getting Creative</u>	1
<u>A Hero Is Born</u>	2
<u>Anatomy Class</u>	4
<u>A La Carte</u>	7
<u>Slice And Dice</u>	10
<u>Killing Off The Kids</u>	12
<u>Rank And File</u>	15
<u>Spider, Spider On The Wall</u>	18
<u>Making Friends And Influencing People</u>	20
<u>Doing The Chameleon</u>	23
<u>Linking Out</u>	26

Getting Creative

If you've been following past issues of this column, you're probably already aware of the numerous things you can do with XML and PHP. You also probably already know the basics – that there are two ways XML can be parsed, either via the Simple API for XML (SAX) or the Document Object Model (DOM); that each method has pros and cons, and that the selection of a parsing technique depends largely on the application; and that PHP today supports both parsing methods, making it a versatile and flexible tool for XML application development.

Now, by default, all newer versions of PHP come with the XML SAX parser enabled; however, the DOM module needs to be explicitly turned on at compile time. If you're working in a development environment that is entirely under your control, this is not a problem – all you need to do is log in as root, recompile PHP and get cracking.

However, if you're working in a shared environment – for example, on your Web hosting provider's server – then it's quite likely that you won't have super-user access to the system. And since Web hosting companies are (understandably!) picky about compiling new software on their production servers, if your default PHP environment doesn't already have DOM support compiled in, it's unlikely that you're going to get it any time soon.

So, a creative solution is needed. Which is where this article comes in.

Over the next few pages, I'm going to be introducing you to a free PHP class named XMLTree, which allows you to manipulate XML document trees in a manner similar (though **not** completely identical) to that available in the standard PHP DOM extension...without requiring you to first recompile your PHP build. As you might imagine, this can come in handy in certain situations – for example, if you need a quick and dirty way to build an XML document tree from an external data source, like a MySQL database or a structured text file. So keep reading – you might find the rest of the show interesting!

A Hero Is Born

The XMLTree class comes courtesy of PEAR, the PHP Extension and Application Repository (<http://pear.php.net>). In case you didn't know, PEAR is an online repository of free PHP software, including classes and modules for everything from data archiving to XML parsing. When you install PHP, a whole bunch of PEAR modules get installed as well; the XMLTree class is one of them.

In case your PHP distribution didn't include XMLTree, you can get yourself a copy from the official PEAR Web site, at <http://pear.php.net> – simply unzip the distribution archive into your PEAR directory and you're ready to roll!

Let's begin with something simple – dynamically constructing an XML document using XMLTree methods. Here's the code:

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("superhero");

// add child elements
$name =& $root->addChild("name", "Peter Parker aka
Spiderman"); $age =&
$root->addChild("age", 21);

// print tree
$tree->dump();

?>
```

Don't worry if it didn't make too much sense – all will be explained shortly. For the moment, just feast your eyes on the output:

```
<?xml version="1.0"?>
```

Building XML Trees With PHP

```
<superhero>  
<name>Peter Parker aka Spiderman</name>  
<age>21</age>  
</superhero>
```

As you can see, the output of the script is a correctly-formatted, well-formed XML document – all created using PHP code!

Anatomy Class

Let's take a closer look at how I accomplished this.

1. The first step is, obviously, to include the XMLTree class

```
<?php

// include class
include("XML/Tree.php");

?>
```

You can either provide an absolute path to this file, or do what most lazy programmers do – include the path to your PEAR installation in PHP's "include_path" variable, so that you can access any of the PEAR classes without needing to type in long, convoluted file paths.

2. Next, an object of the XMLTree class needs to be initialized, and assigned to a PHP variable.

```
<?php

// instantiate object
$tree = new XML_Tree();

?>
```

This variable serves as the control point for future tree manipulation.

3. Every XML document must have a root element – which is where the addRoot() method comes in.

```
<?php

// add the root element
```

```
$root =& $tree->addRoot("superhero");  
  
?>
```

The `addRoot()` method is the starting point for your XML tree – it allows you to specify the name and content of your document's root element, and returns a `Node` object that can be used to graft further branches on to the document tree.

4. The `Node` object returned by the `addRoot()` method comes with methods of its own – and one of the more useful ones is the `addChild()` method, which allows you to add new children under that node. This method is fairly simple to use – all it needs is the name of the child element to be added, and the content and attributes (if any).

```
<?php  
  
// add child elements  
$name =& $root->addChild("name", "Peter Parker aka  
Spiderman"); $age =&  
$root->addChild("age", 21);  
  
?>
```

Each call to `addChild()` returns another `Node` object, which in turn exposes an `addChild()` method, thereby allowing you to add branches to the tree ad infinitum. In this case, I've stopped at a two-level tree – but feel free to go as deep as you like (the example on the next page demonstrates a more complex tree).

5. Once the tree is complete, you can do something useful with it – write it to a file, pass it through a SAX parser or – as I've done here – simply output it to the screen for all to admire.

```
<?php  
  
// print tree  
$tree->dump();  
  
?>
```

The `dump()` method prints the entire XML document tree as is, and serves a very useful purpose in debugging long or complex trees – I'll be using it fairly frequently in the examples in this chapter.

A La Carte

Here's another example, this one building a slightly more complicated document tree,

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("recipe");

// add children
$name =& $root->addChild("name", "Chicken Tikka");
$author =& $root->addChild("author", "Anonymous");
$ingredients =& $root->addChild("ingredients");
$servings =& $root->addChild("servings", 3);
$process =& $root->addChild("process");

// create array of ingredients
$ingredientsArray = array(
    "Boneless chicken breasts, 2",
    "Chopped onions, 2",
    "Garlic, 1 tsp",
    "Ginger, 1 tsp",
    "Red chili powder, 1 tsp",
    "Lime juice, 2 tbsp",
    "Butter, 2 tbsp"
);

// create array of steps to execute recipe
$stepsArray = array(
    "Cut chicken into cubes, wash and apply lime juice and salt",
    "Add ginger, garlic, chili and lime juice in a separate bowl",
    "Mix well, and add chicken to marinate for 3-4 hours",
    "Place chicken pieces on skewers and barbeque",
    "Remove, apply butter, and barbeque again until meat is tender",
    "Garnish with lemon and chopped onions"
);

// add ingredient list to tree
```

```
foreach ($ingredientsArray as $i)
{
$item = $ingredients->addChild("item", $i);
}

// add execution steps to tree
foreach ($stepsArray as $s)
{
$step = $process->addChild("step", $s);
}

// print tree
$tree->dump();

?>
```

and here's the output:

```
<?xml version="1.0"?>

<recipe>

<name>Chicken Tikka</name>

<author>Anonymous</author>

<ingredients>
<item>Boneless chicken breasts, 2</item>
<item>Chopped onions, 2</item>
<item>Garlic, 1 tsp</item>
<item>Ginger, 1 tsp</item>
<item>Red chili powder, 1 tsp</item>
<item>Lime juice, 2 tbsp</item>
<item>Butter, 2 tbsp</item>
</ingredients>

<servings>3</servings>

<process>
<step>Cut chicken into cubes, wash and apply lime juice and
salt</step>
<step>Add ginger, garlic, chili and lime juice in a separate
bowl</step>
<step>Mix well, and add chicken to marinate for 3-4
hours</step>
<step>Place chicken pieces on skewers and barbeque</step>
<step>Remove, apply butter, and barbeque again until meat is
```

```
tender</step>  
<step>Garnish with lemon and chopped onions</step>  
</process>  
</recipe>
```

Slice And Dice

You can retrieve any segment of a dynamically-created document tree via the `get()` method, as in the following variant of the previous example:

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("recipe");

// add children
$name =& $root->addChild("name", "Chicken Tikka");
$author =& $root->addChild("author", "Anonymous");
$ingredients =& $root->addChild("ingredients");
$servings =& $root->addChild("servings", 3);
$process =& $root->addChild("process");

// create array of ingredients
$ingredientsArray = array(
    "Boneless chicken breasts, 2",
    "Chopped onions, 2",
    "Garlic, 1 tsp",
    "Ginger, 1 tsp",
    "Red chili powder, 1 tsp",
    "Lime juice, 2 tbsp",
    "Butter, 2 tbsp"
);

// create array of steps to execute recipe
$stepsArray = array(
    "Cut chicken into cubes, wash and apply lime juice and salt",
    "Add ginger, garlic, chili and lime juice in a separate bowl",
    "Mix well, and add chicken to marinate for 3-4 hours",
    "Place chicken pieces on skewers and barbeque",
    "Remove, apply butter, and barbeque again until meat is tender",
    "Garnish with lemon and chopped onions"
);
```

```
// add ingredient list to tree
foreach ($ingredientsArray as $i)
{
$item = $ingredients->addChild("item", $i);
}

// add execution steps to tree
foreach ($stepsArray as $s)
{
$step = $process->addChild("step", $s);
}

// print ingredients only
echo $ingredients->get();

?>
```

In this case, the output of the script is the <ingredients> node only:

```
<ingredients>
<item>Boneless chicken breasts, 2</item>
<item>Chopped onions, 2</item>
<item>Garlic, 1 tsp</item>
<item>Ginger, 1 tsp</item>
<item>Red chili powder, 1 tsp</item>
<item>Lime juice, 2 tbsp</item>
<item>Butter, 2 tbsp</item>
</ingredients>
```

Killing Off The Kids

You've already seen that you can also add elements to an existing tree via the `addChild()` method. But `XMLTree` also allows you to prune an existing document tree, deleting elements from the hierarchy with the `removeChild()` method.

In order to illustrate this, consider the following example, which dynamically constructs a simple document tree:

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("superhero");

// add child elements
$name =& $root->addChild("name", "Peter Parker aka
Spiderman"); $age =&
$root->addChild("age", 21); $sex =& $root->addChild("sex",
"male");
$location =& $root->addChild("location", "Manhattan");

// print tree
$tree->dump();

?>
```

Here's the output:

```
<?xml version="1.0"?>
<superhero>
<name>Peter Parker aka Spiderman</name>
<age>21</age>
<sex>male</sex>
<location>Manhattan</location>
</superhero>
```

Now's let's suppose I wanted to remove the `<age>` element from the tree. With `removeChild()`, it's a piece of cake:

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("superhero");

// add child elements
$name =& $root->addChild("name", "Peter Parker aka
Spiderman"); $age =&
$root->addChild("age", 21); $sex =& $root->addChild("sex",
"male");
$location =& $root->addChild("location", "Manhattan");

// remove second child node (age)
$root->removeChild(1);

// print tree
$tree->dump();

?>
```

The `removeChild()` method takes, as argument, the index number of the child to be removed under the specified Node object (remember that node numbering begins at 0, not 1). In this case, since I want to remove the `<age>` element, which is the second child of the root element, I can use

```
<?php

$root->removeChild(1);

?>
```

to get the job done.

And here's the revised output:

```
<?xml version="1.0"?>
<superhero>
<name>Peter Parker aka Spiderman</name>
```

```
<sex>male</sex>  
<location>Manhattan</location>  
</superhero>
```

Rank And File

In addition to dynamically creating an XML document tree, XMLTree also allows you to read an existing tree into memory, via its `getTreeFromFile()` method. Consider the following XML file, named "me.xml":

```
<?xml version="1.0"?>
<me>
<name>John Doe</name>
<age>16</age>
<sex>male</sex>
</me>
```

Let's now read this into an XML document tree with the `getTreeFromFile()` method:

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree("me.xml");

// read file contents
$root =& $tree->getTreeFromFile();

// print tree
$tree->dump();

?>
```

In this case, the `getTreeFromFile()` method has been used to read the contents of the XML file specified in the object constructor, and convert it into an XMLTree object. The tree can then be viewed via a call to `dump()`.

Take a quick peek at the object created by `getTreeFromFile()` with the `print_r()` function, and here's what you'll see:

```
xml_tree_node Object
(
  [attributes] => Array
  (
  )

  [children] => Array
```

```
(
[0] => xml_tree_node Object
(
[attributes] => Array
(
)

[children] => Array
(
)

[content] => John Doe
[name] => name
)

[1] => xml_tree_node Object
(
[attributes] => Array
(
)

[children] => Array
(
)

[content] => 16
[name] => age
)

[2] => xml_tree_node Object
(
[attributes] => Array
(
)

[children] => Array
(
)

[content] => male
[name] => sex
)

)

[content] =>
[name] => me
)
```

As you can see, the XML document is converted into a series of nested arrays, each one representing a node on the document tree.

Similar, though not identical, is the `getTreeFromString()` method, which converts an XML-compliant string into an `XMLTree` object. Take a look:

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// create string
$str = "<?xml version='1.0'?> <me> <name>John Doe</name>
<age>16</age>
<sex>male</sex> </me>";

// read string into tree
$root =& $tree->getTreeFromString($str);

// print tree
$tree->dump();

?>
```

Spider, Spider On The Wall...

XMLTree doesn't just restrict you to XML elements – you can easily add attributes to any element as well, simply by specifying them as an associative array of name–value pairs in your calls to the addRoot() or addChild() methods. Here's an example,

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("superhero", NULL, array("owner" =>
"Marvel
Comics"));

// add child elements
$name =& $root->addChild("name", "Peter Parker aka
Spiderman"); $age =&
$root->addChild("age", 21); $sex =& $root->addChild("sex",
"male");
$location =& $root->addChild("location", "Manhattan");

// print tree
$tree->dump();

?>
```

and here's the output:

```
<?xml version="1.0"?>
<superhero owner="Marvel Comics">
<name>Peter Parker aka Spiderman</name>
<age>21</age>
<sex>male</sex>
<location>Manhattan</location>
</superhero>
```

This works with the addChild() method as well – here's an example,

```
<?php

// include class
include("XML/Tree.php");

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("superheroes");

// add child elements
$name =& $root->addChild("member", "Spiderman", array("sex" =>
"male",
"location" => "Manhattan"));

$name =& $root->addChild("member", "Superwoman", array("sex"
=> "female",
"location" => "Metropolis"));

$name =& $root->addChild("member", "Batman", array("sex" =>
"male",
"location" => "Gotham City"));

// print tree
$tree->dump();

?>
```

and here's the XML output.

```
<?xml version="1.0"?>
<superheroes>
<member sex="male" location="Manhattan">Spiderman</member>
<member sex="female" location="Metropolis">Superwoman</member>
<member sex="male" location="Gotham City">Batman</member>
</superheroes>
```



Making Friends And Influencing People

Now, while all this is fine and dandy, how about using all this new-found knowledge for something practical?

This next example does just that, demonstrating how the XMLTree class can be used to convert data stored in a MySQL database into an XML document, and write it to a file for later use. Here's the MySQL table I'll be using,

```
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
--+-----+
| id | name | address | tel
| fax |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
--+-----+
| 1 | Sam Spade | 134, Dark Road, New York, NY |
1-800-TOUGH-GUY | 1-234-587-3636 |
| 2 | The White Rabbit | Down The Rabbit Hole, Wonderland |
56-78-5467
| 56-78-2537 |
| 3 | Jack Lilliput | The Little House, Lilliput Island | None
| None |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
--+-----+
```

and here's what I want my target XML document to look like:

```
<?xml version="1.0"?>
<addressbook>

<item id="1">
<name>Sam Spade</name>
<address>134, Dark Road, New York, NY</address>
<tel>1-800-TOUGH-GUY</tel>
<fax>1-234-587-3636</fax>
</item>

<item id="2">
<name>The White Rabbit</name>
<address>Down The Rabbit Hole, Wonderland</address>
<tel>56-78-5467</tel>
<fax>56-78-2537</fax>
</item>
```

```
<item id="3">
<name>Jack Liliput</name>
<address>The Little House, Lilliput Island</address>
<tel>None</tel>
<fax>None</fax>
</item>

</addressbook>
```

With XMLTree, accomplishing this is a matter of a few lines of code. Here they are:

```
<?php

// include class
include("Tree.php");

// set output filename
$filename = 'addressbook.xml';

// instantiate object
$tree = new XML_Tree();

// add the root element
$root =& $tree->addRoot("addressbook");

// open connection to database
$connection = mysql_connect("localhost", "joe", "secret") or
die ("Unable to
connect!");

// select database
mysql_select_db("db567") or die ("Unable to select
database!");

// execute query
$query = "SELECT * FROM addressbook";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// iterate through rows and print column data
while ($row = mysql_fetch_array($result))
{

// add child elements
$item =& $root->addChild("item", NULL, array("id" =>
```



```
$row['id']));  
$name =& $item->addChild("name", $row['name']);  
$address =& $item->addChild("address", $row['address']);  
$tel =& $item->addChild("tel", $row['tel']);  
$fax =& $item->addChild("fax", $row['fax']);  
  
}  
  
// close database connection  
mysql_close($connection);  
  
// open file  
if (!$handle = fopen($filename, 'w'))  
{  
print "Cannot open file ($filename)";  
exit;  
}  
  
// write XML to file  
if (!fwrite($handle, $tree->get()))  
{  
print "Cannot write to file ($filename)";  
exit;  
}  
  
// close file  
fclose($handle);  
  
?>
```

Pretty simple, once you know how it works. First, I've opened up a connection to the database and retrieved all the records from the table. Then I've instantiated a new document tree and iterated over the result set, adding a new set of nodes to the tree at each iteration. Finally, once all the rows have been processed, the dynamically generated tree is written to a file for later use.



Doing The Chameleon

Why stop there? It's also possible to pair the dynamically-built XML document in the previous example with an XSLT stylesheet to produce HTML output. Here's the stylesheet,

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<head></head>
<body>
<table border="1">
<tr>
<td><b>Name</b></td>
<td><b>Address</b></td>
<td><b>Tel</b></td>
<td><b>Fax</b></td>
</tr>
<xsl:apply-templates/>
</table>
</body>
</html>
</xsl:template>

<xsl:template match="//addressbook/item">
<tr>
<td><xsl:value-of select="name" /></td>
<td><xsl:value-of select="address" /></td>
<td><xsl:value-of select="tel" /></td>
<td><xsl:value-of select="fax" /></td>
</tr>
</xsl:template>

</xsl:stylesheet>
```

and here's a variant of the previous example which, instead of writing the XML to a file, passes it through PHP's XSLT processor to produce an HTML page containing the data from the MySQL database.

```
<?php

// include class
include("Tree.php");
```

```
// set XSLT file
$xmlslt_file = "addressbook.xml";

// instantiate object
$xmltree = new XML_Tree();

// add the root element
$xmlroot =& $xmltree->addRoot("addressbook");

// open connection to database
$xmlconnection = mysql_connect("localhost", "joe", "secret") or
die ("Unable to
connect!");

// select database
mysql_select_db("db567") or die ("Unable to select
database!");

// execute query
$xmlquery = "SELECT * FROM addressbook";
$xmlresult = mysql_query($xmlquery) or die ("Error in query: $xmlquery.
" .
mysql_error());

// iterate through rows and print column data
while ($xmlrow = mysql_fetch_array($xmlresult))
{

// add child elements
$xmlitem =& $xmlroot->addChild("item", NULL, array("id" =>
$xmlrow['id']));
$xmlname =& $xmlitem->addChild("name", $xmlrow['name']);
$xmladdress =& $xmlitem->addChild("address", $xmlrow['address']);
$xmltel =& $xmlitem->addChild("tel", $xmlrow['tel']);
$xmlfax =& $xmlitem->addChild("fax", $xmlrow['fax']);

}

// close database connection
mysql_close($xmlconnection);

// read XML into string
$xml = $xmltree->get();

// create the XSLT processor
$xmlprocessor = xslt_create();

// read in the data
```

Building XML Trees With PHP

```
$xslt = join("", file($xslt_file));

// set up buffers
$args_buffer = array("/xml" => $xml, "/xslt" => $xslt);

// create the XSLT processor
$xml_processor = xslt_create() or die("Could not create XSLT processor");

// process the two strings to get the desired output
if($result = xslt_process($xml_processor, "arg:/xml", "arg:/xslt", NULL,
$args_buffer))
{
echo $result;
}
else
{
echo "An error occurred: " . xslt_error($xml_processor) . "(error code "
.
xslt_errno($xml_processor) . ")";
}

// free the resources occupied by the handler
xslt_free($xml_processor);

?>
```

Here's what the output looks like:

Name	Address	Tel	Fax
Sam Spade	134, Dark Road, New York, NY	1-800-TOUGH-GUY	1-234-587-3636
The White Rabbit	Down The Rabbit Hole, Wonderland	56-78-5467	56-78-2537
Jack Lilliput	The Little House, Lilliput Island	None	None

Simple when you know how!



Linking Out

And that's about it for this article. Over the last few pages, I showed you how you to build an XML document tree even if your PHP build doesn't support the XML DOM, via the free add-on XMLTree class from PEAR. I showed you how to programmatically create an XML document, how to construct root and child nodes and attributes, how to remove nodes from an existing document tree, and how to convert your existing XML files into XMLTree-compliant format Finally, I wrapped things up with a composite example that demonstrated a practical, real-world use for all this code – converting the data in a MySQL database into XML, and transforming it for use on a Web site with PHP and XSLT.

All this is, of course, only the tip of the iceberg – there are an infinite number of possibilities with power like this at your disposal. To find out what else you can do with XML and PHP, I'd encourage you to visit the following links:

XML Basics, at http://www.devshed.com/Server_Side/XML/XMLBasic/

XSL Basics, at http://www.devshed.com/Server_Side/XML/XSLBasics/

Using PHP With XML, at http://www.devshed.com/Server_Side/XML/XMLwithPHP/

XSLT Transformation With PHP And Sablotron, at http://www.devshed.com/Server_Side/XML/XSLTrans

The XML and PHP book, at <http://www.xmlphp.com/>

Till next time...be good!

Note: Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!