# SimpleXML

# Introduction

The SimpleXML extension provides a very simple and easily usable toolset to convert XML to an object that can be processed with normal property selectors and array iterators.

# Installing/Configuring

## Requirements

The SimpleXML extension requires PHP 5.

## Installation

The SimpleXML extension is enabled by default. To disable it, use the *--disable-simplexml* configure option.

## Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

## Resource Types

This extension has no resource types defined.

# Predefined Constants

This extension has no constants defined.

# Examples

Many examples in this reference require an XML string. Instead of repeating this string in every example, we put it into a file which we include in each example. This included file is shown in the following example section. Alternatively, you could create an XML document and read it with simplexml_load_file().

**Example #1 - Include file example.php with XML string**

```
<?php
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<movies>
<movie>
 <title>PHP: Behind the Parser</title>
 <characters>
  <character>
   <name>Ms. Coder</name>
   <actor>Onlivia Actora</actor>
  </character>
  <character>
   <name>Mr. Coder</name>
   <actor>El Act&#211;r</actor>
  </character>
 </characters>
 <plot>
  So, this language. It's like, a programming language. Or is it a
  scripting language? All is revealed in this thrilling horror spoof
  of a documentary.
 </plot>
 <great-lines>
  <line>PHP solves all my web problems</line>
 </great-lines>
 <rating type="thumbs">7</rating>
 <rating type="stars">5</rating>
</movie>
</movies>
XML;
?>
```

The simplicity of SimpleXML appears most clearly when one extracts a string or number from a basic XML document.

**Example #2 - Getting** *<plot>*

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

$xml = new SimpleXMLElement($xmlstr);
```

```
echo $xml->movie[0]->plot; // "So this language. It's like..."
?>
```

Accessing elements within an XML document that contain characters not permitted under PHP's naming convention (e.g. the hyphen) can be accomplished by encapsulating the element name within braces and the apostrophe.

**Example #3 - Getting** *<line>*

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

$xml = new SimpleXMLElement($xmlstr);

echo $xml->movie->{'great-lines'}->line; // "PHP solves all my web problems"
?>
```

**Example #4 - Accessing non-unique elements in SimpleXML**

When multiple instances of an element exist as children of a single parent element, normal iteration techniques apply.

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

$xml = new SimpleXMLElement($xmlstr);

/* For each <movie> node, we echo a separate <plot>. */
foreach ($xml->movie as $movie) {
  echo $movie->plot, '<br />';
}

?>
```

**Example #5 - Using attributes**

So far, we have only covered the work of reading element names and their values. SimpleXML can also access element attributes. Access attributes of an element just as you would elements of an array.

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

$xml = new SimpleXMLElement($xmlstr);

/* Access the <rating> nodes of the first movie.
```

```
* Output the rating scale, too. */
foreach ($xml->movie[0]->rating as $rating) {
   switch((string) $rating['type']) { // Get attributes as element indices
   case 'thumbs':
       echo $rating, ' thumbs up';
       break;
   case 'stars':
       echo $rating, ' stars';
       break;
   }
}
?>
```

**Example #6 - Comparing Elements and Attributes with Text**

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, PHP treats the element as an object.

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

$xml = new SimpleXMLElement($xmlstr);

if ((string) $xml->movie->title == 'PHP: Behind the Parser') {
   print 'My favorite movie.';
}

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

**Example #7 - Using XPath**

SimpleXML includes built-in XPath support. To find all <*character*> elements:

```
<?php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);

foreach ($xml->xpath('//character') as $character) {
   echo $character->name, 'played by ', $character->actor, '<br />';
}
?>
```

' // ' serves as a wildcard. To specify absolute paths, omit one of the slashes.

**Example #8 - Setting values**

Data in SimpleXML doesn't have to be constant. The object allows for manipulation of all of its elements.

```
<?php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);

$xml->movie[0]->characters->character[0]->name = 'Miss Coder';

echo $xml->asXML();
?>
```

The above code will output a new XML document, just like the original, except that the new XML will change Ms. Coder to Miss Coder.

**Example #9 - Adding elements and attributes**

Since PHP 5.1.3, SimpleXML has had the ability to easily add children and attributes.

```
<?php
include 'example.php';
$xml = new SimpleXMLElement($xmlstr);

$character = $xml->movie[0]->characters->addChild('character');
$character->addChild('name', 'Mr. Parser');
$character->addChild('actor', 'John Doe');

$rating = $xml->movie[0]->addChild('rating', 'PG');
$rating->addAttribute('type', 'mpaa');

echo $xml->asXML();
?>
```

The above code will output an XML document based on the original but having a new character and rating.

**Example #10 - DOM Interoperability**

PHP has a mechanism to convert XML nodes between SimpleXML and DOM formats. This example shows how one might change a DOM element to SimpleXML.

```
<?php
$dom = new domDocument;
$dom->loadXML('<books><book><title>blah</title></book></books>');
if (!$dom) {
```

```
    echo 'Error while parsing the document';
    exit;
}

$s = simplexml_import_dom($dom);

echo $s->book[0]->title;
?>
```

# SimpleXML Functions

# SimpleXMLElement->addAttribute()

SimpleXMLElement->addAttribute() -- Adds an attribute to the SimpleXML element

## Description

**SimpleXMLElement**

void **addAttribute** ( string `$name`, string `$value` [, string `$namespace` ] )

Adds an attribute to the SimpleXML element.

## Parameters

*`name`*
: The name of the attribute to add.

*`value`*
: The value of the attribute.

*`namespace`*
: If specified, the namespace to which the attribute belongs.

## Return Values

No value is returned.

## Examples

**Example #11 - Add attributes and children to a SimpleXML element**

```
<?php

include 'example.php';

$sxe = new SimpleXMLElement($xmlstr);
$sxe->addAttribute('type', 'documentary');

$movie = $sxe->addChild('movie');
$movie->addChild('title', 'PHP2: More Parser Stories');
$movie->addChild('plot', 'This is all about the people who make it work.');
```

```
$characters = $movie->addChild('characters');
$character  = $characters->addChild('character');
$character->addChild('name', 'Mr. Parser');
$character->addChild('actor', 'John Doe');

$rating = $movie->addChild('rating', '5');
$rating->addAttribute('type', 'stars');

echo $sxe->asXML();

?>
```

## See Also

- SimpleXMLElement->addChild()

# SimpleXMLElement->addChild()

SimpleXMLElement->addChild() -- Adds a child element to the XML node

## Description

**SimpleXMLElement**

SimpleXMLElement **addChild** ( string `$name` [, string `$value` [, string `$namespace` ] ] )

Adds a child element to the node and returns a SimpleXMLElement of the child.

## Parameters

*`name`*
The name of the child element to add.

*`value`*
If specified, the value of the child element.

*`namespace`*
If specified, the namespace to which the child element belongs.

## Return Values

The *addChild* method returns a SimpleXMLElement object representing the child added to the XML node.

## Examples

**Example #12 - Add attributes and children to a SimpleXML element**

```
<?php

include 'example.php';

$sxe = new SimpleXMLElement($xmlstr);
$sxe->addAttribute('type', 'documentary');

$movie = $sxe->addChild('movie');
$movie->addChild('title', 'PHP2: More Parser Stories');
$movie->addChild('plot', 'This is all about the people who make it work.');
```

```
$characters = $movie->addChild('characters');
$character  = $characters->addChild('character');
$character->addChild('name', 'Mr. Parser');
$character->addChild('actor', 'John Doe');

$rating = $movie->addChild('rating', '5');
$rating->addAttribute('type', 'stars');

echo $sxe->asXML();

?>
```

## See Also

- SimpleXMLElement->addAttribute()

# SimpleXMLElement->asXML()

SimpleXMLElement->asXML() -- Return a well-formed XML string based on SimpleXML element

## Description

**SimpleXMLElement**

mixed **asXML** ( [ string `$filename` ] )

The *asXML* method formats the parent object's data in XML version 1.0.

## Parameters

*`filename`*
If specified, the function writes the data to the file rather than returning it.

## Return Values

If the *`filename`* isn't specified, this function returns a string on success and **FALSE** on error. If the parameter is specified, it returns **TRUE** if the file was written successfully and **FALSE** otherwise.

## Examples

**Example #13 - Get XML**

```
<?php
$string = <<<XML
<a>
<b>
 <c>text</c>
 <c>stuff</c>
</b>
<d>
 <c>code</c>
</d>
</a>
XML;

$xml = new SimpleXMLElement($string);
```

```
echo $xml->asXML(); // <?xml ... <a><b><c>text</c><c>stuff</c> ...

?>
```

*asXML* also works on Xpath results:

**Example #14 - Using asXML() on Xpath results**

```
<?php
// Continued from example XML above.

/* Search for <a><b><c> */
$result = $xml->xpath('/a/b/c');

while(list( , $node) = each($result)) {
   echo $node->asXML(); // <c>text</c> and <c>stuff</c>
}
?>
```

# SimpleXMLElement->attributes()

SimpleXMLElement->attributes() -- Identifies an element's attributes

## Description

**SimpleXMLElement**

SimpleXMLElement **attributes** ( [ string `$ns` [, bool `$is_prefix` ] ] )

This function provides the attributes and values defined within an xml tag.

**Note**

SimpleXML has made a rule of adding iterative properties to most methods. They cannot be viewed using var_dump() or anything else which can examine objects.

## Parameters

*ns*
An optional namespace for the retrieved attributes

*is_prefix*
Default to **FALSE**

## Return Values

## Examples

**Example #15 - Interpret an XML string**

```
<?php
$string = <<<XML
<a xmlns:b>
<foo name="one" game="lonely">1</foo>
</a>
XML;
```

```
$xml = simplexml_load_string($string);
foreach($xml->foo[0]->attributes() as $a => $b) {
   echo $a,'="',$b,"\"\n";
}
?>
```

The above example will output:

```
name="one"
game="lonely"
```

# SimpleXMLElement->children()

SimpleXMLElement->children() -- Finds children of given node

## Description

| SimpleXMLElement |
|---|

SimpleXMLElement **children** ( [ string `$ns` [, bool `$is_prefix` ] ] )

This method finds the children of the element of which it is a member. The result follows normal iteration rules.

| Note |
|---|
| SimpleXML has made a rule of adding iterative properties to most methods. They cannot be viewed using var_dump() or anything else which can examine objects. |

## Parameters

*`ns`*

*`is_prefix`*
Default to **FALSE**

## Return Values

## ChangeLog

| Version | Description |
|---|---|
| 5.2.0 | The optional parameter *`is_prefix`* was added. |

## Examples

**Example #16 - Traversing a *children()* pseudo-array**

```
<?php
$xml = new SimpleXMLElement(
'<person>
<child role="son">
 <child role="daughter"/>
</child>
<child role="daughter">
 <child role="son">
  <child role="son"/>
 </child>
</child>
</person>');

foreach ($xml->children() as $second_gen) {
   echo ' The person begot a ' . $second_gen['role'];

   foreach ($second_gen->children() as $third_gen) {
       echo ' who begot a ' . $third_gen['role'] . ';';

       foreach ($third_gen->children() as $fourth_gen) {
           echo ' and that ' . $third_gen['role'] .
               ' begot a ' . $fourth_gen['role'];
       }
   }
}
?>
```

The above example will output:

```
The person begot a son who begot a daughter; The person
begot a daughter who begot a son; and that son begot a son
```

# SimpleXMLElement->__construct()

SimpleXMLElement->__construct() -- Creates a new SimpleXMLElement object

## Description

**SimpleXMLElement**

**__construct** ( string `$data` [, int `$options` [, bool `$data_is_url` [, string `$ns` [, bool `$is_prefix` ] ] ] ] )

Creates a new SimpleXMLElement object.

## Parameters

*`data`*
A well-formed XML string or the path or URL to an XML document if *`data_is_url`* is **TRUE**.

*`options`*
Optionally used to specify additional Libxml parameters.

*`data_is_url`*
By default, *`data_is_url`* is **FALSE**. Use **TRUE** to specify that *`data`* is a path or URL to an XML document instead of string data.

*`ns`*

*`is_prefix`*

## Return Values

Returns a SimpleXMLElement object representing *`data`*.

## Errors/Exceptions

Produces an **E_WARNING** error message for each error found in the XML data and throws an exception if errors were detected.

## Examples

**Example #17 - Create a SimpleXMLElement object**

```
<?php

include 'example.php';

$sxe = new SimpleXMLElement($xmlstr);
echo $sxe->movie[0]->title;

?>
```

**Example #18 - Create a SimpleXMLElement object from a URL**

```
<?php

$sxe = new SimpleXMLElement('http://example.org/document.xml', NULL, TRUE);
echo $sxe->asXML();

?>
```

## See Also

- simplexml_load_string
- simplexml_load_file

# SimpleXMLElement->getDocNamespaces()

SimpleXMLElement->getDocNamespaces() -- Returns namespaces declared in document

## Description

**SimpleXMLElement**

array **getDocNamespaces** ( [ bool `$recursive` ] )

Returns namespaces declared in document

## Parameters

*`recursive`*
If specified, returns all namespaces declared in parent and child nodes. Otherwise, returns only namespaces declared in root node.

## Return Values

The *getDocNamespaces* method returns an array of namespace names with their associated URIs.

## Examples

**Example #19 - Get document namespaces**

```
<?php

$xml = <<<XML
<?xml version="1.0" standalone="yes"?>
<people xmlns:p="http://example.org/ns">
   <p:person id="1">John Doe</p:person>
   <p:person id="2">Susie Q. Public</p:person>
</people>
XML;

$sxe = new SimpleXMLElement($xml);

$namespaces = $sxe->getDocNamespaces();
var_dump($namespaces);

?>
```

**Example #20 - Working with multiple namespaces**

```
<?php

$xml = <<<XML
<?xml version="1.0" standalone="yes"?>
<people xmlns:p="http://example.org/ns" xmlns:t="http://example.org/test">
   <p:person t:id="1">John Doe</p:person>
   <p:person t:id="2" a:addr="123 Street" xmlns:a="http://example.org/addr">
       Susie Q. Public
   </p:person>
</people>
XML;

$sxe = new SimpleXMLElement($xml);

$namespaces = $sxe->getDocNamespaces(TRUE);
var_dump($namespaces);

?>
```

## See Also

- SimpleXMLElement->getNamespaces()
- SimpleXMLElement->registerXPathNamespace()

# SimpleXMLElement->getName()

SimpleXMLElement->getName() -- Gets the name of the XML element

## Description

**SimpleXMLElement**

string **getName** ( void )

Gets the name of the XML element.

## Return Values

The *getName* method returns as a string the name of the XML tag referenced by the SimpleXMLElement object.

## Examples

**Example #21 - Get XML element names**

```
<?php

$sxe = new SimpleXMLElement($xmlstr);

echo $sxe->getName() . "\n";

foreach ($sxe->children() as $child)
{
   echo $child->getName() . "\n";
}

?>
```

# SimpleXMLElement->getNamespaces()

SimpleXMLElement->getNamespaces() -- Returns namespaces used in document

## Description

**SimpleXMLElement**

array **getNamespaces** ( [ bool `$recursive` ] )

Returns namespaces used in document

## Parameters

*`recursive`*
: If specified, returns all namespaces used in parent and child nodes. Otherwise, returns only namespaces used in root node.

## Return Values

The *getNamespaces* method returns an array of namespace names with their associated URIs.

## Examples

**Example #22 - Get document namespaces in use**

```
<?php

$xml = <<<XML
<?xml version="1.0" standalone="yes"?>
<people xmlns:p="http://example.org/ns" xmlns:t="http://example.org/test">
   <p:person id="1">John Doe</p:person>
   <p:person id="2">Susie Q. Public</p:person>
</people>
XML;

$sxe = new SimpleXMLElement($xml);

$namespaces = $sxe->getNamespaces(true);
var_dump($namespaces);

?>
```

The above example will output:

```
array(1) {
 ["p"]=>
 string(21) "http://example.org/ns"
}
```

## See Also

- SimpleXMLElement->getDocNamespaces()
- SimpleXMLElement->registerXPathNamespace()

# SimpleXMLElement->registerXPathNamespace()

SimpleXMLElement->registerXPathNamespace() -- Creates a prefix/ns context for the next XPath query

## Description

**SimpleXMLElement**

bool **registerXPathNamespace** ( string `$prefix`, string `$ns` )

Creates a prefix/ns context for the next XPath query. In particular, this is helpful if the provider of the given XML document alters the namespace prefixes. *registerXPathNamespace* will create a prefix for the associated namespace, allowing one to access nodes in that namespace without the need to change code to allow for the new prefixes dictated by the provider.

## Parameters

*`prefix`*
The namespace prefix to use in the XPath query for the namespace given in *`ns`*.

*`ns`*
The namespace to use for the XPath query. This must match a namespace in use by the XML document or the XPath query using *`prefix`* will not return any results.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Examples

**Example #23 - Setting a namespace prefix to use in an XPath query**

```
<?php

$xml = <<<EOD
<book xmlns:chap="http://example.org/chapter-title">
   <title>My Book</title>
   <chapter id="1">
       <chap:title>Chapter 1</chap:title>
       <para>Donec velit. Nullam eget tellus vitae tortor gravida
```

```
scelerisque.
           In orci lorem, cursus imperdiet, ultricies non, hendrerit et,
orci.
           Nulla facilisi. Nullam velit nisl, laoreet id, condimentum ut,
           ultricies id, mauris.</para>
   </chapter>
   <chapter id="2">
       <chap:title>Chapter 2</chap:title>
       <para>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin

           gravida. Phasellus tincidunt massa vel urna. Proin adipiscing
quam
           vitae odio. Sed dictum. Ut tincidunt lorem ac lorem. Duis eros
           tellus, pharetra id, faucibus eu, dapibus dictum, odio.</para>
   </chapter>
</book>
EOD;

$sxe = new SimpleXMLElement($xml);

$sxe->registerXPathNamespace('c', 'http://example.org/chapter-title');
$result = $sxe->xpath('//c:title');

foreach ($result as $title) {
 echo $title . "\n";
}

?>
```

Notice how the XML document shown in the example sets a namespace with a prefix of *chap*. Imagine that this document (or another one like it) may have used a prefix of *c* in the past for the same namespace. Since it has changed, the XPath query will no longer return the proper results and the query will require modification. Using *registerXPathNamespace* avoids future modification of the query even if the provider changes the namespace prefix.

## See Also

- SimpleXMLElement->getDocNamespaces()
- SimpleXMLElement->getNamespaces()

# SimpleXMLElement->xpath()

SimpleXMLElement->xpath() -- Runs XPath query on XML data

## Description

**SimpleXMLElement**

array **xpath** ( string `$path` )

The *xpath* method searches the SimpleXML node for children matching the XPath *`path`*.

## Parameters

*`path`*
An XPath path

## Return Values

Returns an array of SimpleXMLElement objects or **FALSE** in case of an error.

## Examples

**Example #24 - Xpath**

```
<?php
$string = <<<XML
<a>
<b>
 <c>text</c>
 <c>stuff</c>
</b>
<d>
 <c>code</c>
</d>
</a>
XML;

$xml = new SimpleXMLElement($string);

/* Search for <a><b><c> */
$result = $xml->xpath('/a/b/c');
```

```
while(list( , $node) = each($result)) {
   echo '/a/b/c: ',$node,"\n";
}

/* Relative paths also work... */
$result = $xml->xpath('b/c');

while(list( , $node) = each($result)) {
   echo 'b/c: ',$node,"\n";
}
?>
```

The above example will output:

```
/a/b/c: text
/a/b/c: stuff
b/c: text
b/c: stuff
```

Notice that the two results are equal.

# simplexml_import_dom

simplexml_import_dom -- Get a *SimpleXMLElement* object from a DOM node.

## Description

SimpleXMLElement **simplexml_import_dom** ( DOMNode `$node` [, string `$class_name` ] )

This function takes a node of a DOM document and makes it into a SimpleXML node. This new object can then be used as a native SimpleXML element.

## Parameters

*`node`*
A DOM Element node

*`class_name`*
You may use this optional parameter so that simplexml_import_dom() will return an object of the specified class. That class should extend the SimpleXMLElement class.

## Return Values

Returns a SimpleXMLElement or **FALSE** on failure.

## Examples

**Example #25 - Importing DOM**

```
<?php
$dom = new DOMDocument;
$dom->loadXML('<books><book><title>blah</title></book></books>');
if (!$dom) {
   echo 'Error while parsing the document';
   exit;
}

$s = simplexml_import_dom($dom);

echo $s->book[0]->title; // blah
?>
```

## See Also

- dom_import_simplexml()

# simplexml_load_file

simplexml_load_file -- Interprets an XML file into an object

## Description

object **simplexml_load_file** ( string `$filename` [, string `$class_name` [, int `$options` [, string `$ns` [, bool `$is_prefix` ] ] ] ] )

Convert the well-formed XML document in the given file to an object.

## Parameters

*`filename`*
Path to the XML file

| **Note** |
| --- |
| Libxml 2 unescapes the URI, so if you want to pass e.g. *b&c* as the URI parameter *a*, you have to call *simplexml_load_file(rawurlencode('http://example.com/?a=' . urlencode('b&c')))*. Since PHP 5.1.0 you don't need to do this because PHP will do it for you. |

*`class_name`*
You may use this optional parameter so that simplexml_load_file() will return an object of the specified class. That class should extend the SimpleXMLElement class.

*`options`*
Since PHP 5.1.0 and Libxml 2.6.0, you may also use the *`options`* parameter to specify additional Libxml parameters.

*`ns`*

*`is_prefix`*

## Return Values

Returns an object of class SimpleXMLElement with properties containing the data held within the XML document. On errors, it will return **FALSE**.

## Examples

**Example #26 - Interpret an XML document**

```
<?php
// The file test.xml contains an XML document with a root element
// and at least an element /[root]/title.

if (file_exists('test.xml')) {
   $xml = simplexml_load_file('test.xml');

   print_r($xml);
} else {
   exit('Failed to open test.xml.');
}
?>
```

This script will display, on success:

```
SimpleXMLElement Object
(
 [title] => Example Title
 ...
)
```

At this point, you can go about using *$xml->title* and any other elements.

## See Also

- simplexml_load_string
- SimpleXMLElement->__construct()

# simplexml_load_string

simplexml_load_string -- Interprets a string of XML into an object

## Description

object **simplexml_load_string** ( string `$data` [, string `$class_name` [, int `$options` [, string `$ns` [, bool `$is_prefix` ] ] ] ] )

Takes a well-formed XML string and returns it as an object.

## Parameters

*`data`*
A well-formed XML string

*`class_name`*
You may use this optional parameter so that simplexml_load_string() will return an object of the specified class. That class should extend the SimpleXMLElement class.

*`options`*
Since PHP 5.1.0 and Libxml 2.6.0, you may also use the *`options`* parameter to specify additional Libxml parameters.

*`ns`*

*`is_prefix`*

## Return Values

Returns an object of class SimpleXMLElement with properties containing the data held within the xml document. On errors, it will return **FALSE**.

## Examples

**Example #27 - Interpret an XML string**

```
<?php
$string = <<<XML
<?xml version='1.0'?>
<document>
<title>Forty What?</title>
<from>Joe</from>
<to>Jane</to>
```

```
<body>
 I know that's the answer -- but what's the question?
</body>
</document>
XML;

$xml = simplexml_load_string($string);

var_dump($xml);
?>
```

The above example will output:

```
SimpleXMLElement Object
(
 [title] => Forty What?
 [from] => Joe
 [to] => Jane
 [body] =>
  I know that's the answer -- but what's the question?
)
```

At this point, you can go about using *$xml->body* and such.

## See Also

- simplexml_load_file
- SimpleXMLElement->__construct()