XSLT

Introduction

This PHP extension provides a processor independent API to XSLT transformations. Currently this extension only supports the Sablotron library from the Ginger Alliance. Support is planned for other libraries, such as the Xalan library or the libxslt library.

XSLT (Extensible Stylesheet Language (XSL) Transformations) is a language for transforming XML documents into other XML documents. It is a standard defined by The World Wide Web Consortium (W3C). Information about XSLT and related technologies can be found at » http://www.w3.org/TR/xslt.

Note
This extension is different than the sablotron extension distributed with versions of PHP prior to PHP 4.1.0, currently only the new XSLT extension in PHP 4.1.0 is supported. If you need support for the old extension, please ask your questions on the PHP mailing lists.

Note
This extension has been moved to the » PECL repository and is no longer bundled with PHP as of PHP 5.0.0.

Note
If you need xslt support with PHP 5 you can use the XSL extension.

Installing/Configuring

Requirements

This extension uses Sablotron and expat, which can both be found at » http://www.gingerall.org/sablotron.html. Binaries are provided as well as source.

Installation

On Unix, run *configure* with the *--enable-xslt --with-xslt-sablot* options. The Sablotron library should be installed somewhere your compiler can find it.

Make sure you have the same libraries linked to the Sablotron library as those, which are linked with PHP. The configuration options: *--with-expat-dir=DIR --with-iconv-dir=DIR* are there to help you specify them. When asking for support, always mention these directives, and whether there are other versions of those libraries installed on your system somewhere. Naturally, provide all the version numbers.

Caution
Be sure your Sablot library is linked to *-lstdc++* as otherwise your configure will fail, or PHP will fail to run or load.

Note
JavaScript E-XSLT support If you compiled Sablotron with JavaScript support, you must specify the option: *--with-sablot-js=DIR*.

Note
Note to Win32 Users In order for this extension to work, there are DLL files that must be available to the Windows system *PATH*. For information on how to do this, see the FAQ entitled " How do I add my PHP directory to the PATH on Windows ". Although copying DLL files from the PHP folder into the Windows system directory also works (because the system directory is by default in the system's *PATH*), this is not recommended. *This extension requires the following files to be in the PATH: sablot.dll*, *expat.dll*, and *iconv.dll* For PHP <= 4.2.0, the file *iconv.dll* is not required.

Runtime Configuration

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

Resource Types

This extension defines a XSLT processor resource returned by xslt_create().

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

XSLT_OPT_SILENT (integer)
Drop all logging and error reporting. This is a generic option for all backends that may be added in the future.

XSLT_SABOPT_PARSE_PUBLIC_ENTITIES (integer)
Tell Sablotron to parse public entities. By default this has been turned off.

XSLT_SABOPT_DISABLE_ADDING_META (integer)
Do not add the meta tag "Content-Type" for HTML output. The default is set during compilation of Sablotron.

XSLT_SABOPT_DISABLE_STRIPPING (integer)
Suppress the whitespace stripping (on data files only).

XSLT_SABOPT_IGNORE_DOC_NOT_FOUND (integer)
Consider unresolved documents (the document() function) non-lethal.

XSLT_SABOPT_FILES_TO_HANDLER (integer)

XSLT_ERR_UNSUPPORTED_SCHEME (integer)
Error return code, for scheme handlers.

XSLT Functions

xslt_backend_info

xslt_backend_info -- Returns the information on the compilation settings of the backend

Description

string **xslt_backend_info** (void)

xslt_backend_info() gets information about the compilation settings of the backend.

Return Values

Returns a string with information about the compilation setting of the backend or an error string when no information available.

See Also

- xslt_backend_name()
- xslt_backend_version()

xslt_backend_name

xslt_backend_name -- Returns the name of the backend

Description

string **xslt_backend_name** (void)

xslt_backend_name() gets the name of the backend.

Return Values

Returns Sablotron.

Examples

Example #1 - xslt_backend_name() example

```
<?php

echo xslt_backend_name(); // Sablotron

?>
```

See Also

- xslt_backend_info()
- xslt_backend_version()

xslt_backend_version

xslt_backend_version -- Returns the version number of Sablotron

Description

string **xslt_backend_version** (void)

xslt_backend_version() gets the version number of Sablotron.

Return Values

Returns the version number, or **FALSE** if not available.

Examples

Example #2 - xslt_backend_version() example

```
<?php

echo xslt_backend_version(); // 0.98 for example

?>
```

See Also

- xslt_backend_info()
- xslt_backend_name()

xslt_create

xslt_create -- Create a new XSLT processor

Description

resource **xslt_create** (void)

Create and return a new XSLT processor resource for manipulation by the other XSLT functions.

Return Values

Returns an XSLT processor link identifier on success, or **FALSE** on error.

Examples

Example #3 - xslt_create() **example**

```
<?php
function xml2html($xmldata, $xsl)
{
   /* $xmldata -> your XML */
   /* $xsl -> XSLT file */

   $path = 'include';
   $arguments = array('/_xml' => $xmldata);
   $xsltproc = xslt_create();
   xslt_set_encoding($xsltproc, 'ISO-8859-1');
   $html =
       xslt_process($xsltproc, 'arg:/_xml', "$path/$xsl", NULL, $arguments);

   if (empty($html)) {
      die('XSLT processing error: '. xslt_error($xsltproc));
   }
   xslt_free($xsltproc);
   return $html;
}
?>
```

See Also

- xslt_free()

xslt_errno

xslt_errno -- Returns an error number

Description

int **xslt_errno** (resource $xh)

Returns an error code describing the last error that occurred on the passed XSLT processor.

Parameters

xh
The XSLT processor link identifier, created with xslt_create().

Return Values

Returns the error code, as an integer.

See Also

- xslt_error()

xslt_error

xslt_error -- Returns an error string

Description

string **xslt_error** (resource `$xh`)

Returns a string describing the last error that occurred on the passed XSLT processor.

Parameters

xh
: The XSLT processor link identifier, created with xslt_create().

Return Values

Returns the error message, as a string.

Examples

Example #4 - Handling errors using the xslt_error() and xslt_errno() functions.

```
<?php

$xh = xslt_create();
$result = xslt_process($xh, 'dog.xml', 'pets.xsl');
if (!$result) {
   die(sprintf("Cannot process XSLT document [%d]: %s",
               xslt_errno($xh), xslt_error($xh)));
}

echo $result;

xslt_free($xh);
?>
```

See Also

- xslt_errno()

xslt_free

xslt_free -- Free XSLT processor

Description

void **xslt_free** (resource `$xh`)

Free the XSLT processor identified by the given handle.

Parameters

xh
: The XSLT processor link identifier, created with xslt_create().

Return Values

No value is returned.

See Also

- xslt_create()

xslt_getopt

xslt_getopt -- Get options on a given xsl processor

Description

int **xslt_getopt** (resource `$processor`)

xslt_getopt() returns the options on the given *`processor`*.

Parameters

`processor`
: The XSLT processor link identifier, created with xslt_create().

Return Values

Returns the options, a bitmask constructed with the *XSLT_SABOPT_XXX* constants.

See Also

- xslt_setopt()

xslt_process

xslt_process -- Perform an XSLT transformation

Description

mixed **xslt_process** (resource `$xh`, string `$xmlcontainer`, string `$xslcontainer` [, string `$resultcontainer` [, array `$arguments` [, array `$parameters`]]])

The xslt_process() function is the crux of the XSLT extension. It allows you to perform an XSLT transformation using almost any type of input source - the containers. This is accomplished through the use of argument buffers -- a concept taken from the Sablotron XSLT processor (currently the only XSLT processor this extension supports). The input containers default to a filename 'containing' the document to be processed.

Parameters

`xh`
: The XSLT processor link identifier, created with xslt_create().

`xmlcontainer`
: Path to XML file or placeholder for the XML argument.

`xslcontainer`
: Path to XSL file or placeholder for the XML argument.

`resultcontainer`
: The result container defaults to a filename for the transformed document. If the result container is not specified - i.e. **NULL** - than the result is returned.

`arguments`
: Instead of files as the XML and XSLT arguments to the xslt_process() function, you can specify "argument place holders" which are then substituted by values given in the *`arguments`* array.

`parameters`
: An array for any top-level parameters that will be passed to the XSLT document. These parameters can then be accessed within your XSL files using the <xsl:param name="parameter_name"> instruction. The parameters must be UTF-8 encoded and their values will be interpreted as strings by the Sablotron processor. In other words - you cannot pass node-sets as parameters to the XSLT document.

Containers can also be set via the *`arguments`* array (see below).

Return Values

Returns **TRUE** on success or **FALSE** on failure. If the result container is not specified - i.e.

NULL - than the result is returned.

ChangeLog

Version	Description
4.0.6	This function no longer takes XML strings in *xmlcontainer* or *xslcontainer*. Passing a string containing XML to either of these parameters will result in a segmentation fault in Sablotron versions up to and including version 0.95.

Examples

The simplest type of transformation with the xslt_process() function is the transformation of an XML file with an XSLT file, placing the result in a third file containing the new XML (or HTML) document. Doing this with sablotron is really quite easy...

Example #5 - Using the xslt_process() to transform an XML file and a XSL file to a new XML file

```
<?php

// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document
if (xslt_process($xh, 'sample.xml', 'sample.xsl', 'result.xml')) {
   echo "SUCCESS, sample.xml was transformed by sample.xsl into result.xml";
   echo ", result.xml has the following contents\n<br />\n";
   echo "<pre>\n";
   readfile('result.xml');
   echo "</pre>\n";
} else {
   echo "Sorry, sample.xml could not be transformed by sample.xsl into";
   echo "  result.xml the reason is that " . xslt_error($xh) . " and the ";
   echo "error code is " . xslt_errno($xh);
}

xslt_free($xh);

?>
```

While this functionality is great, many times, especially in a web environment, you want to be able to print out your results directly. Therefore, if you omit the third argument to the xslt_process() function (or provide a NULL value for the argument), it will automatically return the value of the XSLT transformation, instead of writing it to a file...

Example #6 - Using the xslt_process() to transform an XML file and a XSL file to a variable containing the resulting XML data

```
<?php

// Allocate a new XSLT processor
$xh = xslt_create();

// Process the document, returning the result into the $result variable
$result = xslt_process($xh, 'sample.xml', 'sample.xsl');
if ($result) {
   echo "SUCCESS, sample.xml was transformed by sample.xsl into the
\$result";
   echo " variable, the \$result variable has the following contents\n<br
/>\n";
   echo "<pre>\n";
   echo $result;
   echo "</pre>\n";
} else {
   echo "Sorry, sample.xml could not be transformed by sample.xsl into";
   echo "  the \$result variable the reason is that " . xslt_error($xh);
   echo " and the error code is " . xslt_errno($xh);
}

xslt_free($xh);

?>
```

The above two cases are the two simplest cases there are when it comes to XSLT transformation and I'd dare say that they are the most common cases, however, sometimes you get your XML and XSLT code from external sources, such as a database or a socket. In these cases you'll have the XML and/or XSLT data in a variable -- and in production applications the overhead of dumping these to file may be too much. This is where XSLT's "argument" syntax, comes to the rescue. Instead of files as the XML and XSLT arguments to the xslt_process() function, you can specify "argument place holders" which are then substituted by values given in the arguments array (5th parameter to the xslt_process() function). The following is an example of processing XML and XSLT into a result variable without the use of files at all.

Example #7 - Using the xslt_process() to transform a variable containing XML data and a variable containing XSL data into a variable containing the resulting XML data

```
<?php
// $xml and $xsl contain the XML and XSL data

$arguments = array(
    '/_xml' => $xml,
    '/_xsl' => $xsl
);

// Allocate a new XSLT processor
```

```
$xh = xslt_create();

// Process the document
$result = xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, $arguments);
if ($result) {
   echo "SUCCESS, sample.xml was transformed by sample.xsl into the
\$result";
   echo " variable, the \$result variable has the following contents\n<br
/>\n";
   echo "<pre>\n";
   echo $result;
   echo "</pre>\n";
} else {
   echo "Sorry, sample.xml could not be transformed by sample.xsl into";
   echo "  the \$result variable the reason is that " . xslt_error($xh);
   echo " and the error code is " . xslt_errno($xh);
}
xslt_free($xh);
?>
```

Example #8 - Passing PHP variables to XSL files

```
<?php

// XML string
$xml = '<?xml version="1.0"?>
<para>
change me
</para>';

// XSL string
$xsl = '
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1" indent="no"
omit-xml-declaration="yes"  media-type="text/html"/>
<xsl:param name="myvar"/>
<xsl:param name="mynode"/>
<xsl:template match="/">
My PHP variable : <xsl:value-of select="$myvar"/><br />
My node set : <xsl:value-of select="$mynode"/>
</xsl:template>
</xsl:stylesheet>';


$xh = xslt_create();

// the second parameter will be interpreted as a string
$parameters = array (
 'myvar' => 'test',
 'mynode' => '<foo>bar</foo>'
);

$arguments = array (
 '/_xml' => $xml,
```

```
 '/_xsl' => $xsl
);

echo xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, $arguments,
$parameters);

?>
```

The above example will output:

```
My PHP variable : test<br>
My node set : &lt;foo&gt;bar&lt;/foo&gt;
```

Notes

Note
Please note that *file://* is needed in front of the path when using Windows.

xslt_set_base

xslt_set_base -- Set the base URI for all XSLT transformations

Description

void **xslt_set_base** (resource `$xh`, string `$uri`)

Sets the base URI for all XSLT transformations, the base URI is used with Xpath instructions to resolve document() and other commands which access external resources. It is also used to resolve URIs for the <xsl:include> and <xsl:import> elements.

Parameters

`xh`
The XSLT processor link identifier, created with xslt_create().

`uri`
The base URI to be used.

Return Values

No value is returned.

ChangeLog

Version	Description
4.3.0	As of PHP 4.3.0, the default base URI is the directory of the executing script. In effect, it is the directory name value of the __**FILE**__ constant. The default base URI is less predictable with older versions.

Notes

Note
Please note that *file://* is needed in front of the path when using Windows.

xslt_set_encoding

xslt_set_encoding -- Set the encoding for the parsing of XML documents

Description

void **xslt_set_encoding** (resource `$xh`, string `$encoding`)

Set the output encoding for the XSLT transformations. When using the Sablotron backend, this option is only available when you compile Sablotron with encoding support.

Parameters

`xh`
The XSLT processor link identifier, created with xslt_create().

`encoding`
An output encoding, e.g *iso-8859-1*.

Return Values

No value is returned.

xslt_set_error_handler

xslt_set_error_handler -- Set an error handler for a XSLT processor

Description

void **xslt_set_error_handler** (resource `$xh`, mixed `$handler`)

Set an error handler function for the XSLT processor given by *xh*, this function will be called whenever an error occurs in the XSLT transformation (this function is also called for notices).

Parameters

xh
: The XSLT processor link identifier, created with xslt_create().

handler
: The user function needs to accept four parameters: the XSLT processor, the error level, the error code and an array of messages. The function can be shown as: *error_handler* (resource `$xh`, int `$error_level`, int `$error_code`, array `$messages`)

Return Values

No value is returned.

Examples

Example #9 - xslt_set_error_handler() Example

```
<?php

// Our XSLT error handler
function xslt_error_handler($handler, $errno, $level, $info)
{
 // for now, let's just see the arguments
 var_dump(func_get_args());
}

// XML content :
$xml='<?xml version="1.0"?>
<para>
oops, I misspelled the closing tag
</pata>';

// XSL content :
$xsl='<?xml version="1.0"?>
```

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <strong><xsl:value-of select="para"/></strong>
</xsl:template>
</xsl:stylesheet>';

$xh = xslt_create();

xslt_set_error_handler($xh, "xslt_error_handler");

echo xslt_process($xh, 'arg:/_xml', 'arg:/_xsl',
                 NULL, array("/_xml" => $xml, "/_xsl" => $xsl));

?>
```

The above example will output something similar to:

```
array(4) {
 [0]=>
 resource(1) of type (XSLT Processor)
 [1]=>
 int(3)
 [2]=>
 int(0)
 [3]=>
 array(6) {
   ["msgtype"]=>
   string(5) "error"
   ["code"]=>
   string(1) "2"
   ["module"]=>
   string(9) "Sablotron"
   ["URI"]=>
   string(9) "arg:/_xml"
   ["line"]=>
   string(1) "4"
   ["msg"]=>
   string(34) "XML parser error 7: mismatched tag"
 }
}
```

See Also

- xslt_set_object() if you want to use an object method as handler

xslt_set_log

xslt_set_log -- Set the log file to write log messages to

Description

void **xslt_set_log** (resource `$xh` [, mixed `$log`])

This function allows you to set the file in which you want XSLT log messages to, XSLT log messages are different than error messages, in that log messages are not actually error messages but rather messages related to the state of the XSLT processor. They are useful for debugging XSLT, when something goes wrong.

By default logging is disabled, in order to enable logging you must first call xslt_set_log() with a boolean parameter which enables logging, then if you want to set the log file to debug to, you must then pass it a string containing the filename.

Parameters

xh
The XSLT processor link identifier, created with xslt_create().

log
This parameter is either a boolean value which toggles logging on and off, or a string containing the logfile in which log errors too.

Return Values

No value is returned.

Notes

Note
Please note that *file://* is needed in front of the path when using Windows.

Examples

Example #10 - Using the XSLT Logging features

```
<?php

$xh = xslt_create();
```

```
xslt_set_log($xh, true);
xslt_set_log($xh, getcwd() . '/myfile.log');

$result = xslt_process($xh, 'dog.xml', 'pets.xsl');
echo $result;

xslt_free($xh);
?>
```

xslt_set_object

xslt_set_object -- Sets the object in which to resolve callback functions

Description

bool **xslt_set_object** (resource `$processor`, object `&$obj`)

This function allows to use the *`processor`* inside an *`object`* and to resolve all callback functions in it.

The callback functions can be declared with **xml_set_sax_handlers()**, xslt_set_scheme_handlers() or xslt_set_error_handler() and are assumed to be methods of *`object`*.

Parameters

`processor`
The XSLT processor link identifier, created with xslt_create().

`obj`
An object.

Return Values

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

Examples

Example #11 - Using your own error handler as a method

```
<?php

class my_xslt_processor {

   var $_xh; // our XSLT processor

   function my_xslt_processor()
   {
       $this->_xh = xslt_create();

       // Make $this object the callback resolver
       xslt_set_object($this->_xh, $this);

       // Let's handle the errors
       xslt_set_error_handler($this->_xh, "my_xslt_error_handler");
   }
```

```
    function my_xslt_error_handler($handler, $errno, $level, $info)
    {
        // for now, let's just see the arguments
        var_dump(func_get_args());
    }
}

?>
```

xslt_set_sax_handler

xslt_set_sax_handler -- Set SAX handlers for a XSLT processor

Description

void **xslt_set_sax_handler** (resource `$xh`, array `$handlers`)

Set SAX handlers on the resource handle given by *`xh`*.

Parameters

`xh`
: The XSLT processor link identifier, created with xslt_create().

`handlers`
: SAX handlers should be a two dimensional array with the format (all top level elements are optional):

```
array(
[document] =>
   array(
       start document handler,
       end document handler
   ),
[element] =>
   array(
       start element handler,
       end element handler
   ),
[namespace] =>
   array(
       start namespace handler,
       end namespace handler
   ),
[comment] => comment handler,
[pi] => processing instruction handler,
[character] => character data handler
)
```

Return Values

No value is returned.

xslt_set_sax_handlers

xslt_set_sax_handlers -- Set the SAX handlers to be called when the XML document gets processed

Description

void **xslt_set_sax_handlers** (resource `$processor`, array `$handlers`)

xslt_set_sax_handlers() registers the SAX *`handlers`* for the document, given a XSLT *`processor`* resource.

Using xslt_set_sax_handlers() doesn't look very different than running a SAX parser like xml_parse() on the result of an xslt_process() transformation.

Parameters

`processor`
The XSLT processor link identifier, created with xslt_create().

`handlers`
`handlers` should be an array in the following format:

```
<?php

$handlers = array(

 "document" => array(
   "start_doc",
   "end_doc"),

 "element"  => array(
   "start_element",
   "end_element"),

 "namespace" => array(
   "start_namespace",
   "end_namespace"),

 "comment"   => "comment",

 "pi"        => "pi",

 "character" => "characters"

);
?>
```

Where the functions follow the syntax described for the scheme handler functions.

Note
The given array does not need to contain all of the different sax handler elements (although it can), but it only needs to conform to "handler" => "function" format described above.

Each of the individual SAX handler functions are in the format below:

- *start_doc* (resource `$processor`)
- *end_doc* (resource `$processor`)
- *start_element* (resource `$processor`, string `$name`, array `$attributes`)
- *end_element* (resource `$processor`, string `$name`)
- *start_namespace* (resource `$processor`, string `$prefix`, string `$uri`)
- *end_namespace* (resource `$processor`, string `$prefix`)
- *comment* (resource `$processor`, string `$contents`)
- *pi* (resource `$processor`, string `$target`, string `$contents`)
- *characters* (resource `$processor`, string `$contents`)

Return Values

No value is returned.

Examples

Example #12 - xslt_set_sax_handlers() Example

```
<?php
// From ohlesbeauxjours at yahoo dot fr
// Here's a simple example that applies strtoupper() on
// the content of every <auteur> tag and then displays the
// resulting XML tree:

$xml='<?xml version="1.0"?>
<books>
<book>
 <title>Mme Bovary</title>
 <author>Gustave Flaubert</author>
</book>
<book>
 <title>Mrs Dalloway</title>
 <author>Virginia Woolf</author>
</book>
</books>';

$xsl='<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1" indent="no"
omit-xml-declaration="yes"/>
<xsl:template match="/">
<xsl:for-each select="books/book">
 <livre>
  <auteur><xsl:value-of select="author/text()"/></auteur>
 </livre>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>';

// Handlers :
function start_document()
{
 // start reading the document
}

function end_document()
{
 // end reading the document
}

function start_element($parser, $name, $attributes)
{
 global $result,$tag;
 $result .= "<". $name . ">";
 $tag = $name;
}

function end_element($parser, $name)
{
 global $result;
 $result .= "</" . $name . ">";
}

function characters($parser, $data)
```

```
{
 global $result,$tag;
 if ($tag == "auteur" ) {
   $data = strtoupper($data);
 }
 $result .= $data;
}

// Transformation :
$xh = xslt_create();
$handlers = array("document" => array("start_document","end_document"),
  "element" => array("start_element","end_element"),
  "character" => "characters");

xslt_set_sax_handlers($xh, $handlers);
xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, array("/_xml"=>$xml,
"/_xsl"=>$xsl));
xslt_free($xh);
?>
```

You can also use xslt_set_object() if you want to implement your handlers in an object.

Example #13 - Object oriented handler

```
<?php
// This is the object oriented version of the previous example
class data_sax_handler {

 var $buffer, $tag, $attrs;

 var $_xh;

 function data_sax_handler($xml, $xsl)
 {
     // our xslt resource
     $this->_xh = xslt_create();

     xslt_set_object($this->_xs, $this);

     // configure sax handlers
     $handlers = array(
       "document" => array('start_document', 'end_document'),
       "element" => array('start_element', 'end_element'),
       "character" => 'characters'
     );

     xslt_set_sax_handlers($this->_xh, $handlers);

     xslt_process($this->_xh, 'arg:/_xml', 'arg:/_xsl', NULL,
array("/_xml"=>$xml, "/_xsl"=>$xsl));
     xslt_free($this->_xh);


 }

 function start_document()
 {
```

```
        // start reading the document
 }

 function end_document() {
        // complete reading the document
 }

 function start_element($parser, $name, $attributes) {
        $this->tag = $name;
        $this->buffer .= "<" . $name . ">";
        $this->attrs = $attributes;
 }

 function end_element($parser, $name)
 {
        $this->tag = '';
        $this->buffer .= "</" . $name . ">";
 }

 function characters($parser, $data)
 {
   if ($this->tag == 'auteur') {
          $data = strtoupper($data);
   }
   $this->buffer .= $data;
 }

 function get_buffer() {
   return $this->buffer;
 }

}

$exec = new data_sax_handler($xml, $xsl);

?>
```

Both examples will output:

```
<livre>
  <auteur>GUSTAVE FLAUBERT</auteur>
</livre>
<livre>
  <auteur>VIRGINIA WOOLF</auteur>
</livre>
```

xslt_set_scheme_handler

xslt_set_scheme_handler -- Set Scheme handlers for a XSLT processor

Description

void **xslt_set_scheme_handler** (resource `$xh`, array `$handlers`)

Set Scheme handlers on the resource handle given by *`xh`*.

Parameters

`xh`
: The XSLT processor link identifier, created with xslt_create().

`handlers`
: Scheme handlers should be an array with the format (all elements are optional):

```
array(
[get_all] => get all handler,
[open] => open handler,
[get] => get handler,
[put] => put handler,
[close] => close handler
)
```

Return Values

No value is returned.

xslt_set_scheme_handlers

xslt_set_scheme_handlers -- Set the scheme handlers for the XSLT processor

Description

void **xslt_set_scheme_handlers** (resource `$xh`, array `$handlers`)

Registers the scheme handlers (XPath handlers) for the document.

Parameters

xh
The XSLT processor link identifier, created with xslt_create().

handlers
An array with the following keys: "get_all", "open", "get", "put", and "close". Every entry must be a function name or an array in the following format: array($obj, "method"). Note that the given array does not need to contain all of the different scheme handler elements (although it can), but it only needs to conform to the "handler" => "function" format described above. Each of the individual scheme handler functions called are in the formats below:

```
string   get_all(resource processor, string scheme, string rest)
resource open(resource processor, string scheme, string rest)
int      get(resource processor, resource fp, string &data)
int      put(resource processor, resource fp, string data)
void     close(resource processor, resource fp)
```

Return Values

No value is returned.

Examples

Example #14 - xslt_set_scheme_handlers() example

For example, here is an implementation of the "file_exists()" PHP function.

```
<?php

// Definition of the handler
function mySchemeHandler($processor, $scheme, $rest)
{
   $rest = substr($rest,1);    // to remove the first / automatically added
by the engine
```

```
   if ($scheme == 'file_exists') {
       // result is embedded in a small xml string
       return '<?xml version="1.0" encoding="UTF-8"?><root>' .
(file_exists($rest) ? 'true' : 'false') . '</root>';
   }
}

$SchemeHandlerArray = array('get_all' => 'mySchemeHandler');

// Start the engine
$params = array();
$xh = xslt_create();

xslt_set_scheme_handlers($xh, $SchemeHandlerArray);

$result = xslt_process($xh, "myFile.xml", "myFile.xsl", NULL, array(),
$params);
xslt_free($xh);

echo $result;

?>
```

Then, inside the stylesheet, you can test whether a certain file exists with:

```
<xsl:if test="document('file_exists:anotherXMLfile.xml')/root='true'">
<!-- The file exist -->
</xsl:if>
```

See Also

- xslt_set_scheme_handler()

xslt_setopt

xslt_setopt -- Set options on a given xsl processor

Description

mixed **xslt_setopt** (resource `$processor`, int `$newmask`)

xslt_setopt() sets the options specified by *`newmask`* on the given *`processor`*.

Parameters

`processor`
The XSLT processor link identifier, created with xslt_create().

`newmask`
`newmask` is a bitmask constructed with the following constants:

- **XSLT_SABOPT_PARSE_PUBLIC_ENTITIES** - Tell the processor to parse public entities. By default this has been turned off.
- **XSLT_SABOPT_DISABLE_ADDING_META** - Do not add the meta tag "Content-Type" for HTML output. The default is set during the compilation of the processor.
- **XSLT_SABOPT_DISABLE_STRIPPING** - Suppress the whitespace stripping (on data files only).
- **XSLT_SABOPT_IGNORE_DOC_NOT_FOUND** - Consider unresolved documents (the document() function) non-lethal.

Return Values

Returns the number of previous mask is possible, **TRUE** otherwise, **FALSE** in case of an error.

Examples

Example #15 - xslt_setopt() Example

```
<?php

$xh = xslt_create();

// Tell Sablotron to process public entities
```

```
xslt_setopt($xh, XSLT_SABOPT_PARSE_PUBLIC_ENTITIES);

// Let's also ask him to suppress whitespace stripping
xslt_setopt($xh, xslt_getopt($xh) | XSLT_SABOPT_DISABLE_STRIPPING);

?>
```

See Also

- xslt_getopt()