PHP

- How do I update root certificates in Apache/PHP/cURL environment
- What are the differences between addslashes(), mysql_escape_string() and mysql_real_escape_string()
- PHP and ODBC
- Speed of unpack() in PHP
- Configuring PEAR on Windows
- How do I use cURL in PHP on Windows?
- Passing command-line arguments into PHP
- Using SSL socket in PHP under Windows
- PHP Resources
- PHP error reporting
- PHPXref vs PHPDocumentor
- Create a PHP unit test case using SimpleTest
- PHP Commenting Style
- phpMyAdmin Security
- PHP
- How can I make phpMyAdmin avoid sending MySQL passwords in the clear?
- PHP ODBC Setup Guide
- Performance of array_shift and array_pop in PHP

How do I update root certificates in Apache/PHP/cURL environment

Following is the instruction for dealing with the new ISIS' SSL certificate authority (effective 4/21/2006), Geo Trust, in a UNIX or Windows environment using Apache/PHP/cURL. The instruction can generally apply to any new SSL certificate authority.

UNIX

If your web application is getting an error with ISIS login, try the following:

1. Your PHP was probably compiled with cURL, i.e, —with-curl=/usr/local/curl-7.12.0. Our cURL is installed in /usr/local/curl-7.12.0, but yours can be any arbitary path. Find out what is it.

2. Your cURL came with the default CA bundle file, which contains root certificates for all the well known certificate authorities at the time cURL was installed. Our file is /usr/local/curl-7.12.0/share/curl/curl-ca-bundle.crt, which is the default location for the default compilation of cURL. If you compiled cURL with a custom location for this file, find out what is it and that's the one you will update.

3. Looked for the new ISIS certificate authority from Geo Trust in /usr/local/curl-7.12.0/share/curl/curl-ca-bundle.crt. Basically all the following 3 lines should be in curl-ca-bundle.crt:

Equifax Secure Global eBusiness CA-1
Validity Period: Mon Jun 21, 1999 to Sun Jun 21, 2020 (GMT)
Certificate Fingerprint (MD5): 8F:5D:77:06:27:C4:98:3C:5B:93:78:E7:D7:7D:9B:CC

If any of these lines are not in curl-ca-bundle.crt, you need to update your curl-ca-bundle.crt.

4a. If you don't have any local certificates in curl-ca-bundle.crt, you can replace the entire curl-ca-bundle.crt. Save the old curl-ca-bundle.crt and get cacert.pem from

http://curl.haxx.se/docs/caextract.html. Replace curl-ca-bundle.crt with cacert.pem.

4b. If you have some local certificates in curl-ca-bundle.crt, get cacert.pem from

http://curl.haxx.se/docs/caextract.html and extract "Equifax Secure Global eBusiness CA" certificate from cacert.pem by extracting the lines between and including:

Equifax Secure Global eBusiness CA

and

END CERTIFICATE

Make a copy of the current curl-ca-bundle.crt and then append this piece of new certificate data to curl-ca-bundle.crt.

5. Restart your Apache server (the PHP module in Apache reads curl-ca-bundle.crt at startup).

6. Test login to ISIS.

Windows

cURL in Apache/PHP on Windows doesn’t read a CA Bundle at startup and must be set by the application. On Windows adjust your CA Bundle file as above for UNIX. If you don’t have one already read this.

What are the differences between addslashes(), mysql_escape_string() and mysql_real_escape_string()

addslashes() escapes single quote (’), double quote ("), backslash (\) and NUL (\x00).

mysql_escape_string() and mysql_real_escape_string() escapes the characters above plus: CR (\r), LF (\n) and EOF (\x1a). Apparently (according to the manual), MySQL wants these characters escaped too, but my experiment shows otherwise (i.e. MySQL doesn’t care if these characters are in a string).

Suppose:

```
$value = 'bar'; // 'ba' and then CR-LF and then 'r'
```

print “insert into pairs values (‘foo’, ’” . addslashes($value) . “’)” gives:

```
insert into pairs values ('foo', 'ba\r\nr')
```

print “insert into pairs values (‘foo’, ’” . mysql_real_escape_string($value) . “’)” gives:

```
insert into pairs values ('foo', 'bar')
```

In this case, the execution result should be the same, but the statement itself is different.

For other EOF, the execution result and statement are identical for both functions.

mysql_real_escape_string() is available on PHP 4.3.0 or above. mysql_escape_string() is deprecated and you should use mysql_real_escape_string() instead, as it takes the current character set into account when escaping characters.

addslashes() should be enough for single-byte strings. For multi-byte strings though, mysql_real_escape_string() does provide better security. See this article for details.

PHP manual on:

- addslashes

- mysql_escape_string
- mysql_real_escape_string

PHP and ODBC

While looking for something else in the Moodle Forums, I found these links that refer to the underlying way Moodle connects to databases using ODBC.

“adodb just harnesses the underlying PHP functions for whatever type of connection you use, so it helps to be familiar with how they work.”

- http://uk.php.net/odbc
- http://www.phpfreaks.com/tutorials/61/0.php
- http://phplens.com/phpeverywhere/node/view/9
- http://bryanmills.net:8086/archives/2003/11/microsoft-access-database-using-linux-and-php/

(Links taken from this Moodle Forums post. http://moodle.org/mod/forum/discuss.php?d=74133)

- https://kb.ucla.edu/link/1088

Please add more, if you find any.

Speed of unpack() in PHP

I needed to extract a list of integers from a binary string. I was curious to know if PHP’s unpack function is fast compared to a custom function, so I wrote a test for it.

```
<?php$filename = '<some large file>';$unpack_time = 0.0;$unpack_time2 = 0.0;$unpack_custom_time
```

The printout: (The numbers are the number of seconds needed to convert the file’s content into integers. The file I used was 2.67 MB in size.)

```
$unpack_time = 1.34654474258$unpack_time2 = 1.44259476662$unpack_custom_time = 127.910765171
```

The result matches my earlier experiment, from which I have learned that custom functions are much slower than PHP’s built-in ones. Whenever possible, use the latter. By do that, you also have less code to write and debug.

Configuring PEAR on Windows

PEAR is the PHP Extension and Application Repository. Applications written in PHP often include references to external libraries and PEAR is a way to manage these. On Windows if PEAR hasn't already been configured log in to the server and run go-pear.bat in the PHP directory (C:\php in this case). This adds the PEAR settings to php.ini:

```
include_path=".;C:\php\pear"
```

It also creates "pear.bat" which can be used to search for and install components. For instance:

```
pear search Requestpear install HTTP_Request
```

If you use the Apache Web Server after running go-pear.bat it's necessary to restart Apache (as with any other changes to php.ini).

How do I use cURL in PHP on Windows?

To configure cURL to be able to run in PHP uncomment this line (remove the semi-colon) in the php.ini file:

;extension=php_curl.dll
Apparently in UNIX systems Apache will read cURL's curl-ca-bundle.crt file at startup and cURL will be able to use that information. The regular Windows Apache version does not have a full cURL installation, merely the .dll (as referenced above). It will not read curl-ca-bundle.crt in the folder with php_curl.dll and it will also not read curl-ca-bundle.crt in Apache's configuration folder. To get this functionality under Windows in your application you must set the CURLOPT_CAINFO option to point to the location of a Certificate Authority Bundle file like this:

curl_setopt($ch, CURLOPT_CAINFO, 'C:/accessible/by/apache/cacert.pem);
Once this is done you will be able to verify SSL certificates by setting the VERIFYPEER option to true (the default for later versions of cURL) like this:

curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, true);

A good CA Bundle file can be found on this page. If you have problems you may need to update your CA Bundle file. In particular, the SSL Certificate for the Test ISIS Server cannot be read using curl-ca-bundle.crt from the latest full cURL version for Windows.

Passing command-line arguments into PHP

Say you have a PHP script and you want to pass command-line arguments into the script, e.g. calling the script like this:

```
php script.php datafile.txt 10 100
```

PHP stores all command-line arguments in an array:
$argv [0] => “script.php”
$argv [1] => “datafile.txt”
$argv [2] => 10
$argv [3] => 100

Then you can process the arguments:

```
if (!isset($argv[1]) {
print “Usage: php script.php  [ ]\n”;
exit(1);
}
```

$filename = $argv[1];

...

See the article Using PHP from the command line for details.

Using SSL socket in PHP under Windows

Problem

If you try to open a socket (fsockopen, pfsockopen) with SSL in PHP 4.x under Windows, the operation might fail with the following message: `Warning:  fsockopen(): no SSL support in this build`

This problem occurs even if phpinfo() shows openssl as loaded and the command `php -m` shows openssl as one of the loaded module.

Cause

To use SSL in sockets, PHP core must be compiled with OpenSSL, which is not the case with the binary available at php.net. On the other hand, the openssl module only enables the OpenSSL functions (functions prefixed openssl_). According to the bug report linked below, “OpenSSL support enabled” that phpinfo() states just means that the openssl **extension** is available.

Solution

This problem has been identified and labeled “won’t fix” in the offical PHP distribution (details). You have the following options:

- Upgrade to PHP 5.x, which does not have such problem.
- If you are using PHP 4.3.x, you can download a (unofficial) SSL-enabled php4ts.dll.
- Use PHP’s curl extension (which is supported by the Windows binary from php.net) instead.

PHP Resources

PHP is a web programming language that can be compiled into the Apache web server and with its persistent connections to MySQL it makes for a very popular and fast web programming environment.

- PHP Cheat Sheet
  - https://websitesetup.org/php-cheat-sheet/
- Documenting PHP
  - phpDocumentator
  - phpXref
- Joseph Vaughan's favorite articles on PHP security
  - http://www.linuxjournal.com/article.php?sid=6061
  - http://www.linuxjournal.com/article.php?sid=6559
- Use @ to turn off warning error message on a given command
- MVC frameworks for PHP (things like Ruby-on-Rails)
  - Laravel: https://laravel.com
  - Cake: http://www.cakephp.org
  - Symfony: http://www.symfony-project.com
  - CodeIgniter: www.codeigniter.com
- Debian packages for bleeding-edge versions of PHP and other LAMP-related packages: http://www.dotdeb.org
- PHPunit – Unit Testing for PHP
- http://codepad.org/ – test snippets of code in PHP and many other languages.

PHP error reporting

Error levels

name value description example 1 example 2 E_ERROR 1 Fatal run-time errors notdefined(); E_WARNING 2 Run-time warnings 1 / 0; E_PARSE 4 Compile-time parse errors +-; E_eval('+-;'); E_NOTICE 8 Run-time notices print $i_am_not_defined; E_CORE_* varies generated by PHP core E_USER_* varies generated by trigger_error E_ALL 2047 Everything

Custom Error Handler

It can only catch E_WARNING, E_PARSE, E_USER_ERROR, E_USER_WARNING and E_USER_NOTICE. If something else happens, PHP's default error handler takes place.

```
functional my_error_handler($errno, $error, $file, $line) {  ... handle the error ...}
```

set_error_handler('my_error_handler');

Apache

In Apache, you can customize how it responses to a particular HTTP status code. It can show a message or load a specific page. e.g. (from Apache's doc)

```
ErrorDocument 500 /cgi-bin/crash-recoverErrorDocument 500 "Sorry, our script crashed. Oh dearErr
http://xxx/
ErrorDocument 404 /Lame_excuses/not_found.htmlErrorDocument 401 /Subscription/how_to_subscribe.h
```

However, when executed as an Apache module PHP returns a HTTP status code of 200 (OK) even if there is an compile or run-time error. (Not sure if it's a bug or a feature, since this behavior is not mentioned anywhere else.) Therefore, one cannot use Apache's custom error capability for PHP errors.

Instead, either use a custom error handler (see above) or have PHP wrap the message inside some specific HTML code (see below).

Pretty-printing Error Messages

Config keys error_prepend_string and error_append_string are used to determine the HTML code that prepends or appends an error message.

e.g. To display each error message inside a box, put these in php.ini:

```
; String to output before an error message.error_prepend_string = "<fieldset>An error has occure
admin@yourname.com
\">administrator</a> with the following error message: <br>"; String to output after an error me
```

To make the change effective to a directory instead of system-wide, put these in .htaccess:

```
php_value error_prepend_string "(... starting tags ...)"php_value error_append_string "(... endi
```

Note: Because any compile-time error stops the compilation, there will be at most one box.

PHPXref vs PHPDocumentor

Introduction

There exists a wide array of PHP documentation tools on the web available free for download. Two of the most popular ones are PHPXref and PHPDocumentor. Here, I outline the differences between the two to help you decide which one you should use.

The following information was taken from http://phpxref.sourceforge.net/ and http://www.phpdoc.org/.

Similarities

The greatest similarity between the two tools is, of course, their ability to read PHP documents and output information about them in another format, namely HTML. Both of these documentors highlights elements of the source in order to increase readability. PHPXref has also adopted the PHPdoc commenting standard to give additional information about pages, while recent versions of PHPdocumentor has adopted PHPXref's ability to cross reference source code.

The Strengths of PHPXref

1. PHPXref is programmed in Perl which is directly run by your machine. In that way, it is potentially faster than PHPdocumentor which does the processing through PHP itself.
2. The HTML output of PHPXref is very neat – it implements a bit of javascript to provide neat rollover visuals on additional information.
3. Perhaps the strongest feature of PHPXref is its ability to provide information even without PHPdoc comments.
4. Very easy to use and set up. You simply drag the files you want information about in a source folder, run the command prompt, and grab the documentation from the source folder.
5. Although I haven't tested it, PHPXref mentions that they provide information about SQL tables.

The Strengths of PHPDocumentor

1. Provides both a web and a command line interface.
2. Can output in CHM, HTML, and *PDF*!!
3. The interface allows *a lot* more customization than PHPXref. It can generate the resultant documentation in your own web interface (or you can choose one of their pre-built ones) in addition to choosing the source directory and the output directory. This level of customization is really what makes PHPXref more difficult to use than PHPXref which decides those things for you.

4. Has an active community with a well-detailed online documentation.
   #Can create class inheritance diagrams (Not tested)
5. Their documentation IS the standard. Expect new features and updates.
6. Bundled with Zend Studio, a popular web development software.

Summary

Perhaps the main reason to use PHPXref is its ability to provide information without PHPdoc comments. This ability gives PHPXref a speed advantage if you want quick documentation without having to go back and comment lines and lines of code (although you should consider getting around to it!). This type of development favors procedural programming to an extent.

Personally, I like PHPdocumentor's ability to create PDF's and its online documentation. Another cool feature is perhaps the ability to infer class inheritance straight from the source code. The rest of the features of PHPdocumentor is just eye candy.

Keep in mind that PHPdocumentor will have immediate support for PHPdoc style commenting. New versions of PHP may create a need for more commenting standards that PHPdocumentor will be quick to adopt.

I suspect that in the future, PHPdocumentor will continue to adopt useful features from other documentation tools. But in reality, the major differences between PHPXref and PHPdocumentor have been neutralized – using either one will likely suffice for your needs.

Create a PHP unit test case using SimpleTest

You can download SimpleTest at https://sourceforge.net/projects/simpletest/

Suppose you have a PHP file called `math.php` that contains functions that you want to test.

```
<?function square($x) {□return $x * $x;}function cube($x) {□return $x * $x * $x;}?>
```

Then you can write a test case in another file, say `mathtest.php`.

(Suppose you have downloaded and extracted SimpleTest to `C:\simpletest`.)

```
<?phpif (!defined('SIMPLE_TEST')) {□define('SIMPLE_TEST', 'C:\\simpletest\\');}require_once(SIMP
```

(Each function named test* in this class represents a test case and will be run automatically by SimpleTest.)

Run the test case with the command `php mathtest.php`. As the `square` function is written correctly, the test case will pass.

```
testofloggingOKTest cases run: 1/1, Passes: 1, Failures: 0, Exceptions: 0
```

Try changing the `square` function and see what happens.

PHP Commenting Style

Any programmer can tell you that good commenting in your source code is an integral part of programming. Whether the language you're dealing with is an interpreted language like Javascript or compiled like C++, good comments lead to better readability and better flow of logic.

Developers of the Java programming language have come up with a strict commenting standard for Java to interface with a program called Javadoc. Not only does this standard remind the programmer to make appropriate comments, it also allows Javadoc to parse the strict comments into HTML documentation. The success of Javadoc carried over PHP with the creation of PHPDoc. The commenting syntax of PHPDoc is widely supported in the PHP community. Parsers for this standard include phpxref and phpdocumentor.

Basic Syntax

Here's an example of PHPDoc style comments:

```
/*** This is a short description** This is a longer description of the element that* I will comm
```

All PHPDoc comments begin with /** on its own line at the top and ends with a */ at the bottom. Each line of commenting is denoted by the single asterik at the beginning of each line. By convetion, PHPDoc comments begin with a short description followed by a longer description. Let's look at a more realistic example where we wish to comment on a dispenseIceCream() function:

```
/*** Dispenses desired flavored ice cream** Called upon when user has entered their* favorite ic
```

I've thrown a lot of new things in the above code, but the syntax should be somewhat self-explanatory. First, you should have noticed the @ tags. Tags in PHPDoc tell the parser exactly what you are going to be describing in your comments. Notice that each tag has its own parameters, separated by a space as in the @param tag. Tags are, in essence, the strength of PHPDoc.

@access tells the parser that the following element we are commenting on (the dispenseIceCream function) is a private function. @param gives the parser information about the parameters that the function takes in. Finally, @return provides information about what kind of output to expect out of the function.

Each tag in PHPDoc has its own syntax. For more information, look at the documentation

Useful Tags

Using **every single tag** in PHPDoc leads to the issue of overcommenting and obscurity of the actual code! Here, I outline what I deem to be useful tags to place in your document (tag info from phpdocumentor):

@access (private | protected | public)
The access tag will allow the documentation to clearly state how an element should be accessed. This is important for programming design purposes.

@param datatype $paramname description
I consider this tag to be one the most important tags in PHPDoc. It allows a quick look at how the input of a function should be formatted in order to use it.

@return datatype description
Similar to the @param tag, when dealing with functions, one of the things a programmer cares to look for is what this function will output.

@uses (please refer to documentation for proper syntax)
This tag allows actual linking and backlinking to another element! Its a neat feature that allows someone browsing the documentation to quickly see related functions or variables the element uses. The syntax is a bit tricky, so please study it.

@var datatype description
A nice tag to comment on important variables.

Other Tags to Note

The following tags might be useful or nice in your commenting and documentation, but I wouldn't consider them to be the bread and butter of PHPDoc.

@global datatype $globalvariablename | description
Global variables should be given more attention to. This tag should be used with @var.

@ignore
For whatever reason, you may not want PHPDoc to parse the following element. the @ignore tag (with no parameters) does just that. You can go ahead and give it a description if you feel its necessary.

@static
Similar to @access, program design is one of the things programmers may want to reference in your code. The static tag whether a a certain class of method should be treated as static.

@staticvar
Same as @static, except for variables.

inline {@link URL description}
This in-line tag is very useful whenever you wish to reference something outside within your descriptions. It creates a link to the URL you specify within the documentation. There are other inline tags, but this one is the most versatile (but possibly not the easiest to use).

phpMyAdmin Security

- phpMyAdmin Security Announcements

PHP

- **PHP**
  - eXtremePHP and http://pear.php.net/ *These code bases will be useful in rolling the PHP out more quickly – Jose*
  - A higher level PHP web application framework is the Horde framework http://www.horde.org/ – Jose
  - http://www.phpoki.org/

- CakePHP
- Zend
- Symfony
- CodeIgniter
- PHP on Trax
- Seagull

How can I make phpMyAdmin avoid sending MySQL passwords in the clear?

Although phpMyAdmin is an excellent tool for administering MySQL databases, you don’t want to expose your MySQL usernames and passwords to sniffing over the wire by sending them “in the clear.”

The solution, if you are running https, is to simple edit the config.inc.php file like this. The default is FALSE.

$cfg[‘ForceSSL’] = TRUE; // whether to force using https

PHP ODBC Setup Guide

Guide to setting up php-odbc for connection to Registrar database. Example for RedHat EL 5.

Create file tds.datasource.template:

```
[Registrar]
Driver  = FreeTDS
Description  = UCLA Registrar (SRDB)
Trace  = No
Server  = srdb.registrar.ucla.edu
Port  = 1433
```

Create file tds.datasource.template:

```
[FreeTDS]
Description     = v0.63 with protocol v8.0
Driver          = /usr/lib/libtdsodbc.so
```

Make sure Driver points to an existing file.

Run the following command:
sudo yum install php-odbc
sudo odbcinst -i -d -f tds.driver.template
odbcinst -i -s -f tds.datasource.template
sudo cp ~/.odbc.ini /etc/odbc.ini

Test the connection:

```
<?
$conn = odbc_connect('Registrar', $username, $password);
$result = odbc_exec($conn, "EXECUTE CIS_facultyCourseStudentsGetAlpha2 '254049110', '081'");
?>
```

Note on programming: do not nest odbc_exec() calls! Make one call, copy records into array, odbc_free_result() it, and then move on. Second odbc_exec() will fail otherwise.

More info on settings at http://www.unixodbc.org/doc/FreeTDS.html

Performance of array_shift and array_pop in PHP

We have confirmed that array_shift is much slower than array_pop in PHP.

Code:

```
<?
```

```
// Create an array with 100000 elements

$array = array();
for ($i = 0; $i < 100000; $i++) {
$array[] = rand();
}

// Remove the last 1000 elements using array_pop

$start = microtime(true);
for ($i = 0; $i < 1000; $i++) {
array_pop($array);
}
$stop = microtime(true);
printf("array_pop takes %.5f seconds\n", $stop – $start);

// Add back 1000 elements

for ($i = 0; $i < 1000; $i++) {
$array[] = rand();
}

// Remove the first 1000 elements using array_shift

$start = microtime(true);
for ($i = 0; $i < 1000; $i++) {
array_shift($array);
}
$stop = microtime(true);
printf("array_shift takes %.5f seconds\n", $stop – $start);

// Add back 1000 elements
```

```
for ($i = 0; $i < 1000; $i++) {
$array[] = rand();
}

// Remove the first 1000 elements by reversing the array and popping 1000 elements

$start = microtime(true);
$array_rev = array_reverse($array);
for ($i = 0; $i < 1000; $i++) {
array_pop($array_rev);
}
$stop = microtime(true);
printf("array_reverse + array_pop takes %.5f seconds\n", $stop – $start);

?>
```

Result:

```
array_pop takes 0.00089 seconds
array_shift takes 15.15544 seconds
array_reverse + array_pop takes 0.03934 seconds
```