PHP Manual

Stig Sæther Bakken
Alexander Aulbach
Egon Schmid
Jim Winstead
Lars Torben Wilson
Rasmus Lerdorf
Andrei Zmievski
Jouni Ahto

Edited by

Stig Sæther Bakken
Egon Schmid

27-01-2003

Copyright © 1997, 1998, 1999, 2000, 2001, 2002, 2003 the PHP Documentation Group

Copyright

This manual is © Copyright 1997 - 2003 by the PHP Documentation Group. The members of this group are listed on the front page of this manual.

This manual can be redistributed under the terms of the GNU General Public License as published by the Free Software Foundation: either version 2 of the License, or (at your option) any later version.

The 'Extending PHP 4.0' section of this manual is copyright © 2000 by Zend Technologies, Ltd. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

Table of Contents

Preface

> PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Its syntax draws upon C, Java, and Perl, and is easy to learn. The main goal of the language is to allow web developers to write dynamically generated webpages quickly, but you can do much more with PHP.

This manual consists primarily of a function reference, but also contains a language reference, explanations of some of PHP's major features, and other supplemental information.

You can download this manual in several formats at http://www.php.net/docs.php. The downloads are updated as the content changes. More information about how this manual is developed can be found in the 'About the manual' appendix.

See also PHP History

I. Getting Started

Table of Contents

Chapter 1. Introduction

What is PHP?

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

Simple answer, but what does that mean? An example:

Example 1-1. An introductory example

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
```

```
    <title>Example</title>
  </head>
  <body>

    <?php
    echo "Hi, I'm a PHP script!";
    ?>

  </body>
</html>
```

Notice how this is different from a script written in other languages like Perl or C -- instead of writing a program with lots of commands to output HTML, you write an HTML script with some embedded code to do something (in this case, output some text). The PHP code is enclosed in special start and end tags that allow you to jump into and out of "PHP mode".

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server. If you were to have a script similar to the above on your server, the client would receive the results of running that script, with no way of determining what the underlying code may be. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve.

The best things in using PHP are that it is extremely simple for a newcomer, but offers many advanced features for a professional programmer. Don't be afraid reading the long list of PHP's features. You can jump in, in a short time, and start writing simple scripts in a few hours.

Although PHP's development is focused on server-side scripting, you can do much more with it. Read on, and see more in the What can PHP do? section.

What can PHP do?

Anything. PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. But PHP can do much more.

There are three main fields where PHP scripts are used.

- Server-side scripting. This is the most traditional and main target field for PHP. You need three things to make this work. The PHP parser (CGI or server module), a webserver and a web browser. You need to run the webserver, with a connected PHP installation. You can access the PHP program output with a web browser, viewing the PHP page through the server. See the installation instructions section for more information.
- Command line scripting. You can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way. This type of usage is ideal for scripts regularly executed using cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks. See the section about Command line usage of PHP for more information.
- Writing client-side GUI applications. PHP is probably not the very best language to write windowing applications, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can also use PHP-GTK to write such programs. You also have the ability to write cross-platform applications this way. PHP-GTK is an extension to PHP, not available in the main distribution. If you are interested in PHP-GTK, visit it's own website.

PHP can be used on all major operating systems, including Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X, RISC OS, and probably others. PHP has also support for most of the web servers today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, and many others. For the majority of the servers PHP has a module, for the others supporting the CGI standard, PHP can work as a CGI processor.

So with PHP, you have the freedom of choosing an operating system and a web server. Furthermore, you also have the choice of using procedural programming or object oriented programming, or a mixture of them. Although not every standard OOP feature is realized in the current version of PHP, many code libraries and large applications (including the PEAR library) are written only using OOP code.

With PHP you are not limited to output HTML. PHP's abilities includes outputting images, PDF files and even Flash movies (using libswf and Ming) generated on the fly. You can also output easily any text, such as XHTML and any other XML file. PHP can autogenerate these files, and save them in the file system, instead of printing it out, forming a server-side cache for your dynamic content.

One of the strongest and most significant feature in PHP is its support for a wide range of databases. Writing a database-enabled web page is incredibly simple. The following databases are currently supported:

Adabas D	Ingres	Oracle (OCI7 and OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (read-only)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

We also have a DBX database abstraction extension allowing you to transparently use any database supported by that extension. Additionally PHP supports ODBC, the Open Database Connection standard, so you can connect to any other database supporting this world standard.

PHP also has support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (on Windows) and countless others. You can also open raw network sockets and interact using any other protocol. PHP has support for the WDDX complex data exchange between virtually all Web programming languages. Talking about interconnection, PHP has support for instantiation of Java objects and using them transparently as PHP objects. You can also use our CORBA extension to access remote objects.

PHP has extremely useful text processing features, from the POSIX Extended or Perl regular expressions to parsing XML documents. For parsing and accessing XML documents, we support the SAX and DOM standards. You can use our XSLT extension to transform XML documents.

While using PHP in the ecommerce field, you'll find the Cybercash payment, CyberMUT, VeriSign Payflow Pro and CCVS functions useful for your online payment programs.

At last but not least, we have many other interesting extensions, the mnoGoSearch search engine functions, the IRC Gateway functions, many compression utilities (gzip, bz2), calendar conversion, translation...

As you can see this page is not enough to list all the features and benefits PHP can offer. Read on in the sections about installing PHP, and see the function reference part for explanation of the extensions mentioned here.

As you can see this page is not enough to list all the features and benefits PHP can offer. Read on in the sections about installing PHP, and see the function reference part for explanation of the extensions mentioned here.

Chapter 2. A simple tutorial

Here we would like to show the very basics of PHP in a short simple tutorial. This text only deals with dynamic webpage creation with PHP, though PHP is not only capable of creating webpages. See the section titled What can PHP do for more information.

PHP-enabled web pages are treated just like regular HTML pages and you can create and edit them the same way you normally create regular HTML pages.

What do I need?

In this tutorial we assume that your server has support for PHP activated and that all files ending in .php are handled by PHP. On most servers this is the default extension for PHP files, but ask your server administrator to be sure. If your server supports PHP then you don't need to do anything. Just create your .php files and put them in your web directory and the server will magically parse them for you. There is no need to compile anything nor do you need to install any extra tools. Think of these PHP-enabled files as simple HTML files with a whole new family of magical tags that let you do all sorts of things.

Let's say you want to save precious bandwidth and develop locally. In this case, you'll want to install a web server, such as Apache, and of course PHP. You'll most likely want to install a database as well, such as MySQL. You can install these individually or a simpler way is to locate a pre-configured package that automatically installs all of these with just a few mouse clicks. It's easy to setup a web server with PHP support on any operating system, including Linux and Windows. In linux, you may find rpmfind helpful for locating RPMs.

Your first PHP-enabled page

Create a file named hello.php and put it in your web servers root directory (DOCUMENT_ROOT) with the following content:

Example 2-1. Our first PHP script: hello.php

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo "<p>Hello World</p>"; ?>
 </body>
</html>
```

Use your browser to access the file with your web access URL, ending with the "/hello.php" file reference. When developing locally this url will be something like http://localhost/hello.php or http://127.0.0.1/hello.php but this depends on the web servers configuration. Although this is outside the scope of this tutorial, see also the DocumentRoot and ServerName directives in your web servers configuration file. (on Apache this is httpd.conf). If everything is setup correctly, this file will be parsed by PHP and the following output will make it to your browser:

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <p>Hello World</p>
 </body>
</html>
```

Note that this is not like a CGI script. The file does not need to be executable or special in any way. Think of it as a normal HTML file which happens to have a set of special tags available to you that do a lot of interesting things.

This program is extremely simple and you really didn't need to use PHP to create a page like this. All it does is display: Hello World using the PHP **echo()** statement.

If you tried this example and it didn't output anything, or it prompted for download, or you see the whole file as text, chances are that the server you are on does not have PHP enabled. Ask your administrator to enable it for you using the Installation chapter of the manual. If you're developing locally, also read the installation chapter to make sure everything is configured properly. If problems continue to persist, don't hesitate to use one of the many PHP support options.

The point of the example is to show the special PHP tag format. In this example we used <?php to indicate the start of a PHP tag. Then we put the PHP statement and left PHP mode by adding the closing tag, ?>. You may jump in and out of PHP mode in an HTML file like this all you want. For more details, read the manual section on basic PHP syntax.

> **A Note on Text Editors:** There are many text editors and Integrated Development Environments (IDEs) that you can use to create, edit and manage PHP files. A partial list of these tools is maintained at PHP Editor's List. If you wish to recommend an editor, please visit the above page and ask the page maintainer to add the editor to the list. Having an editor with syntax highlighting can be helpful.
>
> **A Note on Word Processors:** Word processors such as StarOffice Writer, Microsoft Word and Abiword are not good choices for editing PHP files. If you wish to use one for this test script, you must ensure that you save the file as PLAIN TEXT or PHP will not be able to read and execute the script.
>
> **A Note on Windows Notepad:** If you are writing your PHP scripts using Windows Notepad, you will need to ensure that your files are saved with the .php extension. (Notepad adds a .txt extension to files automatically unless you take one of the following steps to prevent it.) When you save the file and are prompted to provide a name for the file, place the filename in quotes (i.e. "hello.php"). Alternately, you can click on the 'Text Documents' drop-down menu in the save dialog box and change the setting to "All Files". You can then enter your filename without quotes.

Now that you've successfully created a simple PHP script that works, it's time to create the most famous PHP script! Make a call to the **phpinfo()** function and you'll see a lot of useful information about your system and setup such as available Predefined Variables, loaded PHP modules, and configuration settings. Take some time and review this important information.

Something Useful

Something Useful

Let's do something a bit more useful now. We are going to check what sort of browser the person viewing the page is using. In order to do that we check the user agent string that the browser sends as part of its HTTP request. This information is stored in a variable. Variables always start with a dollar-sign in PHP. The variable we are interested in right now is $_SERVER["HTTP_USER_AGENT"].

> **PHP Autoglobals Note:** $_SERVER is a special reserved PHP variable that contains all web server information. It's known as an Autoglobal (or Superglobal). See the related manual page on Autoglobals for more information. These special variables were introduced in PHP 4.1.0. Before this time, we used the older $HTTP_*_VARS arrays instead, such as $HTTP_SERVER_VARS. Although deprecated, these older variables still exist. (See also the note on old code.)

To display this variable, we can simply do:

Example 2-2. Printing a variable (Array element)

```
<?php echo $_SERVER["HTTP_USER_AGENT"]; ?>
```

A sample output of this script may be:

```
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```

There are many types of variables available in PHP. In the above example we printed an Array element. Arrays can be very useful.

$_SERVER is just one variable that's automatically made available to you by PHP. A list can be seen in the Reserved Variables section of the manual or you can get a complete list of them by creating a file that looks like this:

Example 2-3. Show all predefined variables with phpinfo()

```
<?php phpinfo(); ?>
```

If you load up this file in your browser you will see a page full of information about PHP along with a list of all the variables available to you.

You can put multiple PHP statements inside a PHP tag and create little blocks of code that do more than just a single echo. For example, if we wanted to check for Internet Explorer we could do something like this:

Example 2-4. Example using control structures and functions

```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
    echo "You are using Internet Explorer<br />";
}
?>
```

A sample output of this script may be:

```
You are using Internet Explorer<br />
```

Here we introduce a couple of new concepts. We have an if statement. If you are familiar with the basic syntax used by the C language this should look logical to you. If you don't know enough C or some other language where the syntax used above is used, you should probably pick up any introductory PHP book and read the first couple of chapters, or read the Language Reference part of the manual. You can find a list of PHP books at http://www.php.net/books.php.

The second concept we introduced was the **strstr()** function call. **strstr()** is a function built into PHP which searches a string for another string. In this case we are looking for "MSIE" inside $_SERVER["HTTP_USER_AGENT"]. If the string is found, the function returns **TRUE** and if it isn't, it returns **FALSE**. If it returns **TRUE**, the if statement evaluates to **TRUE** and the code within its {braces} is executed. Otherwise, it's not. Feel free to create similar examples, with if, else, and other functions such as **strtoupper()** and **strlen()**. Each related manual page contains examples too.

We can take this a step further and show how you can jump in and out of PHP mode even in the middle of a PHP block:

Example 2-5. Mixing both HTML and PHP modes

```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
?>
<h3>strstr must have returned true</h3>
<center><b>You are using Internet Explorer</b></center>
<?php
} else {
?>
<h3>strstr must have returned false</h3>
<center><b>You are not using Internet Explorer</b></center>
<?php
}
?>
```

A sample output of this script may be:

```
<h3>strstr must have returned true</h3>
<center><b>You are using Internet Explorer</b></center>
```

Instead of using a PHP echo statement to output something, we jumped out of PHP mode and just sent straight HTML. The important and powerful point to note here is that the logical flow of the script remains intact. Only one of the HTML blocks will end up getting sent to the viewer depending on if **strstr()** returned **TRUE** or **FALSE** In other words, if the string MSIE was found or not.

Dealing with Forms

Dealing with Forms

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element in a form will automatically be available to your PHP scripts. Please read the manual section on Variables from outside of PHP for more information and examples on using forms with PHP. Here's an example HTML form:

Example 2-6. A simple HTML form

```
<form action="action.php" method="POST">
 Your name: <input type="text" name="name" />
 Your age: <input type="text" name="age" />
 <input type="submit">
</form>
```

There is nothing special about this form. It is a straight HTML form with no special tags of any kind. When the user fills in this form and hits the submit button, the action.php page is called. In this file you would have something like this:

Example 2-7. Printing data from our form

```
Hi <?php echo $_POST["name"]; ?>.
You are <?php echo $_POST["age"]; ?> years old.
```

A sample output of this script may be:

```
Hi Joe.
You are 22 years old.
```

It should be obvious what this does. There is nothing more to it. The $_POST["name"] and $_POST["age"] variables are automatically set for you by PHP. Earlier we used the $_SERVER autoglobal, now above we just introduced the $_POST autoglobal which contains all POST data. Notice how the method of our form is POST. If we used the method GET then our form information would live in the $_GET autoglobal instead. You may also use the $_REQUEST autoglobal if you don't care the source of your request data. It contains a mix of GET, POST, COOKIE and FILE data. See also the **import_request_variables()** function.

Using old code with new versions of PHP

Now that PHP has grown to be a popular scripting language, there are more resources out there that have listings of code you can reuse in your own scripts. For the most part the developers of the PHP language have tried to be backwards compatible, so a script written for an older version should run (ideally) without changes in a newer version of PHP, in practice some changes will usually be needed.

Two of the most important recent changes that affect old code are:

- The deprecation of the old $HTTP_*_VARS arrays (which need to be indicated as global when used inside a function or method). The following autoglobal arrays were introduced in PHP 4.1.0. They are: $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV, $_REQUEST, and $_SESSION. The older $HTTP_*_VARS arrays, such as $HTTP_POST_VARS, still exist and have since PHP 3.
- External variables are no longer registered in the global scope by default. In other words, as of PHP 4.2.0 the PHP directive register_globals is off by default in php.ini. The preferred method of accessing these values is via the autoglobal arrays mentioned above. Older scripts, books, and tutorials may rely on this directive being on. If on, for example, one could use $id from the URL http://www.example.com/foo.php?id=42. Whether on or off, $_GET['id'] is available.

For more details on these changes, see the section on predefined variables and links therein.

What's next?

With what you know now you should be able to understand most of the manual and also the various example scripts available in the example archives. You can also find other examples on the php.net websites in the links section: http://www.php.net/links.php.

Chapter 3. Installation

General Installation Considerations

Before installing first, you need to know what do you want to use PHP for. There are three main fields you can use PHP, as described in the What can PHP do? section:

- Server-side scripting
- Command line scripting
- Client-side GUI applications

For the first and most common form, you need three things: PHP itself, a web server and a web browser. You probably already have a web browser, and depending on your operating system setup, you may also have a web server (eg. Apache on Linux or IIS on Windows). You may also rent webspace at a company. This way, you don't need to set up anything on your own, only write your PHP scripts, upload it to the server you rent, and see the results in your browser.

While setting up the server and PHP on your own, you have two choices for the method of connecting PHP to the server. For many servers PHP has a direct module interface (also called SAPI). These servers include Apache, Microsoft Internet Information Server, Netscape and iPlanet servers. Many other servers have support for ISAPI, the Microsoft module interface (OmniHTTPd for example). If PHP has no module support for your web server, you can always use it as a CGI processor. This means you set up your server to use the command line executable of PHP (php.exe on Windows) to process all PHP file requests on the server.

If you are also interested to use PHP for command line scripting (eg. write scripts autogenerating some images for you offline, or processing text files depending on some arguments you pass to them), you always need the command line executable. For more information, read the section about writing command line PHP applications. In this case, you need no server and no browser.

With PHP you can also write client side GUI applications using the PHP-GTK extension. This is a completely different approach than writing web pages, as you do not output any HTML, but manage windows and objects within them. For more information about PHP-GTK, please visit the site dedicated to this extension. PHP-GTK is not included in the official PHP distribution.

With PHP you can also write client side GUI applications using the PHP-GTK extension. This is a completely different approach than writing web pages, as you do not output any HTML, but manage windows and objects within them. For more information about PHP-GTK, please visit the site dedicated to this extension. PHP-GTK is not included in the official PHP distribution.

From now on, this section deals with setting up PHP for web servers on Unix and Windows with server module interfaces and CGI executables.

Downloading PHP, the source code, and binary distributions for Windows can be found at http://www.php.net/. We recommend you to choose a mirror nearest to you for downloading the distributions.

Unix/HP-UX installs

This section contains notes and hints specific to installing PHP on HP-UX systems.

Example 3-1. Installation Instructions for HP-UX 10

From: paul_mckay@clearwater-it.co.uk
04-Jan-2001 09:49
(These tips are for PHP 4.0.4 and Apache v1.3.9)

So you want to install PHP and Apache on a HP-UX 10.20 box?

1. You need gzip, download a binary distribution from
http://hpux.connect.org.uk/ftp/hpux/Gnu/gzip-1.2.4a/gzip-1.2.4a-sd-10.20.depot.Z
uncompress the file and install using swinstall

2. You need gcc, download a binary distribution from
http://gatekeep.cs.utah.edu/ftp/hpux/Gnu/gcc-2.95.2/gcc-2.95.2-sd-10.20.depot.gz
gunzip this file and install gcc using swinstall.

3. You need the GNU binutils, you can download a binary distribution from
http://hpux.connect.org.uk/ftp/hpux/Gnu/binutils-2.9.1/binutils-2.9.1-sd-10.20.depot.gz
gunzip and install using swinstall.

4. You now need bison, you can download a binary distribution from
http://hpux.connect.org.uk/ftp/hpux/Gnu/bison-1.28/bison-1.28-sd-10.20.depot.gz
install as above.

5. You now need flex, you need to download the source from one of the
http://www.gnu.org mirrors. It is in the <filename>non-gnu</filename> directory of the ftp site.
Download the file, gunzip, then tar -xvf it. Go into the newly created flex
directory and do a ./configure, then a make, and then a make install

If you have errors here, it's probably because gcc etc. are not in your
PATH so add them to your PATH.

Right, now into the hard stuff.

6. Download the PHP and apache sources.

7. gunzip and tar -xvf them.

We need to hack a couple of files so that they can compile ok.

8. Firstly the configure file needs to be hacked because it seems to lose
track of the fact that you are a hpux machine, there will be a
better way of doing this but a cheap and cheerful hack is to put
    lt_target=hpux10.20
on line 47286 of the configure script.

9. Next, the Apache GuessOS file needs to be hacked. Under
apache_1.3.9/src/helpers change line 89 from
    "echo "hp${HPUXMACH}-hpux${HPUXVER}"; exit 0"
to:
    "echo "hp${HPUXMACH}-hp-hpux${HPUXVER}"; exit 0"

10. You cannot install PHP as a shared object under HP-UX so you must compile
it as a static, just follow the instructions at the Apache page.

11. PHP and apache should have compiled OK, but Apache won't start. you need
to create a new user for Apache, eg www, or apache. You then change lines 252
and 253 of the conf/httpd.conf in Apache so that instead of
    User nobody
    Group nogroup
you have something like
    User www
    Group sys

This is because you can't run Apache as nobody under hp-ux.
Apache and PHP should then work.

Hope this helps somebody,
Paul Mckay.

Unix/Linux installs

This section contains notes and hints specific to installing PHP on Linux distributions.

Using Packages

Many Linux distributions have some sort of package installation system, such as RPM. This can assist in setting up a standard configuration, but if you need to have a different set of features (such as a secure server, or a different database driver), you may need to build PHP and/or your webserver. If you are unfamiliar with building and compiling your own software, it is worth

Many Linux distributions have some sort of package installation system, such as RPM. This can assist in setting up a standard configuration, but if you need to have a different set of features (such as a secure server, or a different database driver), you may need to build PHP and/or your webserver. If you are unfamiliar with building and compiling your own software, it is worth checking to see whether somebody has already built a packaged version of PHP with the features you need.

Unix/Mac OS X installs

This section contains notes and hints specific to installing PHP on Mac OS X Server.

Using Packages

There are a few pre-packaged and pre-compiled versions of PHP for Mac OS X. This can help in setting up a standard configuration, but if you need to have a different set of features (such as a secure server, or a different database driver), you may need to build PHP and/or your web server yourself. If you are unfamiliar with building and compiling your own software, it's worth checking whether somebody has already built a packaged version of PHP with the features you need.

Compiling for OS X server

There are two slightly different versions of Mac OS X, client and server. The following is for OS X Server.

Example 3-2. Mac OS X server install

1. Get the latest distributions of Apache and PHP
2. Untar them, and run the configure program on Apache like so.
   ./configure --exec-prefix=/usr \
   --localstatedir=/var \
   --mandir=/usr/share/man \
   --libexecdir=/System/Library/Apache/Modules \
   --iconsdir=/System/Library/Apache/Icons \
   --includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
   --enable-shared=max \
   --enable-module=most \
   --target=apache

4. You may also want to add this line:
   setenv OPTIM=-O2
   If you want the compiler to do some optimization.

5. Next, go to the PHP 4 source directory and configure it.
   ./configure --prefix=/usr \
   --sysconfdir=/etc \
   --localstatedir=/var \
   --mandir=/usr/share/man \
   --with-xml \
   --with-apache=/src/apache_1.3.12

   If you have any other additions (MySQL, GD, etc.), be sure to add
   them here. For the --with-apache string, put in the path to your
   apache source directory, for example "/src/apache_1.3.12".
6. make
7. make install
   This will add a directory to your Apache source directory under
   src/modules/php4.

8. Now, reconfigure Apache to build in PHP 4.
   ./configure --exec-prefix=/usr \
   --localstatedir=/var \
   --mandir=/usr/share/man \
   --libexecdir=/System/Library/Apache/Modules \
   --iconsdir=/System/Library/Apache/Icons \
   --includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
   --enable-shared=max \
   --enable-module=most \
   --target=apache \
   --activate-module=src/modules/php4/libphp4.a

   You may get a message telling you that libmodphp4.a is out of date.
   If so, go to the src/modules/php4 directory inside your apache
   source directory and run this command:

   ranlib libmodphp4.a

   Then go back to the root of the apache source directory and run the
   above configure command again. That'll bring the link table up to
   date.

9. make

10. make install

11. copy and rename the php.ini-dist file to your "bin" directory from your
    PHP 4 source directory:
    cp php.ini-dist /usr/local/bin/php.ini

    or (if your don't have a local directory)

    cp php.ini-dist /usr/bin/php.ini

Compiling for MacOS X client

Those tips are graciously provided by Marc Liyanage.

Those tips are graciously provided by Marc Liyanage.

The PHP module for the Apache web server included in Mac OS X. This version includes support for the MySQL and PostgreSQL databases.

NOTE: Be careful when you do this, you could screw up your Apache web server!

Do this to install:

- 1. Open a terminal window
- 2. Type "wget http://www.diax.ch/users/liyanage/software/macosx/libphp4.so.gz", wait for download to finish
- 3. Type "gunzip libphp4.so.gz"
- 4. Type "sudo apxs -i -a -n php4 libphp4.so"

Now type "sudo open -a TextEdit /etc/httpd/httpd.conf" TextEdit will open with the web server configuration file. Locate these two lines towards the end of the file: (Use the Find command)

```
#AddType application/x-httpd-php .php
  #AddType application/x-httpd-php-source .phps
```

Remove the two hash marks (#), then save the file and quit TextEdit.

Finally, type "sudo apachectl graceful" to restart the web server.

PHP should now be up and running. You can test it by dropping a file into your "Sites" folder which is called "test.php". Into that file, write this line: "<?php phpinfo() ?>".

Now open up 127.0.0.1/~your_username/test.php in your web browser. You should see a status table with information about the PHP module.

Unix/OpenBSD installs

This section contains notes and hints specific to installing PHP on OpenBSD 3.2.

Using Binary Packages

Using binary packages to install PHP on OpenBSD is the recommended and simplest method. The core package has been separated from the various modules, and each can be installed and removed independently from the others. The files you need can be found on your OpenBSD CD or on the FTP site.

The main package you need to install is php4-core-4.2.3.tgz, which contains the basic engine (plus gettext and iconv). Next, take a look at the module packages, such as php4-mysql-4.2.3.tgz or php4-imap-4.2.3.tgz. You need to use the **phpxs** command to activate and deactivate these modules in your php.ini file.

Example 3-3. OpenBSD Package Install Example

```
# pkg_add php4-core-4.2.3.tgz
# /usr/local/sbin/phpxs -s
# cp /usr/local/share/doc/php4/php.ini-recommended /var/www/conf/php.ini
  (add in mysql)
# pkg_add php4-mysql-4.2.3.tgz
# /usr/local/sbin/phpxs -a mysql
  (add in imap)
# pkg_add php4-imap-4.2.3.tgz
# /usr/local/sbin/phpxs -a imap
  (remove mysql as a test)
# pkg_delete php4-mysql-4.2.3
# /usr/local/sbin/phpxs -r mysql
  (install the PEAR libraries)
# pkg_add php4-pear-4.2.3.tgz
```

Read the packages(7) manual page for more information about binary packages on OpenBSD.

Using Ports

You can also compile up PHP from source using the ports tree. However, this is only recommended for users familiar with OpenBSD. The PHP4 port is split into three sub-directories: core, extensions and pear. The extensions directory generates sub-packages for all of the supported PHP modules. If you find you do not want to create some of these modules, use the **no_*** FLAVOR. For example, to skip building the imap module, set the FLAVOR to **no_imap**.

Older Releases

Older releases of OpenBSD used the FLAVORS system to compile up a statically linked PHP. Since it is hard to generate binary packages using this method, it is now deprecated. You can still use the old stable ports trees if you wish, but they are unsupported by the OpenBSD team. If you have any comments about this, the current maintainer for the port is Anil Madhavapeddy.

Unix/Solaris installs

This section contains notes and hints specific to installing PHP on Solaris systems.

Required software

Solaris installs often lack C compilers and their related tools. The required software is as follows:

- gcc (recommended, other C compilers may work)
- make
- flex
- bison
- m4
- autoconf
- automake
- perl

- autoconf
- automake
- perl
- gzip
- tar
- GNU sed

In addition, you will need to install (and possibly compile) any additional software specific to your configuration, such as Oracle or MySQL.

Using Packages

You can simplify the Solaris install process by using pkgadd to install most of your needed components.

Installation on UNIX systems

This section will guide you through the general configuration and installation of PHP on Unix systems. Be sure to investigate any sections specific to your platform or web server before you begin the process.

Prerequisite knowledge and software:

- Basic UNIX skills (being able to operate "make" and a C compiler, if compiling)
- An ANSI C compiler (if compiling)
- flex (for compiling)
- bison (for compiling)
- A web server
- Any module specific components (such as gd, pdf libs, etc.)

There are several ways to install PHP for the Unix platform, either with a compile and configure process, or through various pre-packaged methods. This documentation is mainly focused around the process of compiling and configuring PHP.

The initial PHP setup and configuration process is controlled by the use of the commandline options of the configure script. This page outlines the usage of the most common options, but there are many others to play with. Check out the Complete list of configure options for an exhaustive rundown. There are several ways to install PHP:

- As an Apache module
- As an fhttpd module
- For use with AOLServer, NSAPI, phttpd, Pi3Web, Roxen, thttpd, or Zeus.
- As a CGI executable

Apache Module Quick Reference

PHP can be compiled in a number of different ways, but one of the most popular is as an Apache module. The following is a quick installation overview.

Example 3-4. Quick Installation Instructions for PHP 4 (Apache Module Version)

1. gunzip apache_1.3.x.tar.gz
2. tar xvf apache_1.3.x.tar
3. gunzip php-x.x.x.tar.gz
4. tar xvf php-x.x.x.tar
5. cd apache_1.3.x
6. ./configure --prefix=/www
7. cd ../php-x.x.x
8. ./configure --with-mysql --with-apache=../apache_1.3.x --enable-track-vars
9. make
10. make install
11. cd ../apache_1.3.x
12. ./configure --activate-module=src/modules/php4/libphp4.a
13. make
14. make install
15. cd ../php-x.x.x
16. cp php.ini-dist /usr/local/lib/php.ini
17. Edit your httpd.conf or srm.conf file and add:
    AddType application/x-httpd-php .php
18. Use your normal procedure for restarting the Apache server. (You must stop and restart the server, not just cause the server to reload by use a HUP or USR1 signal.)

Building

When PHP is configured, you are ready to build the CGI executable. The command **make** should take care of this. If it fails and you can't figure out why, see the Problems section.

Installation on Windows systems

This section applies to Windows 95/98/Me and Windows NT/2000/XP. Do not expect PHP to work on 16 bit platforms such as Windows 3.1. Sometimes we refer to the supported Windows platforms as Win32.

There are two main ways to install PHP for Windows: either manually or by using the InstallShield installer.

If you have Microsoft Visual Studio, you can also build PHP from the original source code.

Once you have PHP installed on your Windows system, you may also want to load various extensions for added functionality.

Windows InstallShield

The Windows PHP installer available from the downloads page at http://www.php.net/downloads.php, this installs the CGI versionof PHP and, for IIS, PWS, and Xitami, configures the web server as well.

The Windows PHP installer available from the downloads page at http://www.php.net/downloads.php, this installs the CGI versionof PHP and, for IIS, PWS, and Xitami, configures the web server as well.

> **Note:** Also note, that while the InstallShield installer is an easy way to make PHP work, it is restricted in many aspects, as automatic setup of extensions for example is not supported. The whole set of supported extensions is only available by downloading the zip binary distribution.

Install your selected HTTP server on your system and make sure that it works.

Run the executable installer and follow the instructions provided by the installation wizard. Two types of installation are supported - standard, which provides sensible defaults for all the settings it can, and advanced, which asks questions as it goes along.

The installation wizard gathers enough information to set up the php.ini file and configure the web server to use PHP. For IIS and also PWS on NT Workstation, a list of all the nodes on the server with script map settings is displayed, and you can choose those nodes to which you wish to add the PHP script mappings.

Once the installation has completed the installer will inform you if you need to restart your system, restart the server, or just start using PHP.

Warning
Be aware, that this setup of PHP is not secure. If you would like to have a secure PHP setup, you'd better go on the manual way, and set every option carefully. This automatically working setup gives you an instantly working PHP installation, but it is not meant to be used on online servers.

Manual Installation Steps

This install guide will help you manually install and configure PHP on your Windows webserver. You need to download the zip binary distribution from the downloads page at http://www.php.net/downloads.php. The original version of this guide was compiled by Bob Silva, and can be found at http://www.umesd.k12.or.us/php/win32install.html.

This guide provides manual installation support for:

- Personal Web Server 3 and 4 or newer
- Internet Information Server 3 and 4 or newer
- Apache 1.3.x
- OmniHTTPd 2.0b1 and up
- Oreilly Website Pro
- Xitami
- Netscape Enterprise Server, iPlanet

PHP 4 for Windows comes in two flavours - a CGI executable (php.exe), and several SAPI modules (for example: php4isapi.dll). The latter form is new to PHP 4, and provides significantly improved performance and some new functionality. There is also a CLI version which is further described in the commandline chapter.

Warning
The SAPI modules have been significantly improved in the 4.1 release, however, you may find that you encounter possible server errors or other server modules such as ASP failing, in older systems.

If you choose one of the SAPI modules and use Windows 95, be sure to download the DCOM update from the Microsoft DCOM pages. For the ISAPI module, an ISAPI 4.0 compliant Web server is required (tested on IIS 4.0, PWS 4.0 and IIS 5.0). IIS 3.0 is NOT supported. You should download and install the Windows NT 4.0 Option Pack with IIS 4.0 if you want native PHP support.

The following steps should be performed on all installations before the server specific instructions.

- Extract the distribution file to a directory of your choice. c:\php\ is a good start. You probably do not want to use a path in which spaces are included (for example: c:\program files\php is not a good idea). Some web servers will crash if you do.
- You need to ensure that the DLLs which PHP uses can be found. The precise DLLs involved depend on which web server you use and whether you want to run PHP as a CGI or as a server module. php4ts.dll is always used. If you are using a server module (e.g. ISAPI or Apache) then you will need the relevant DLL from the sapi folder. If you are using any PHP extension DLLs then you will need those as well. To make sure that the DLLs can be found, you can either copy them to the system directory (e.g. winnt/system32 or windows/system) or you can make sure that they live in the same directory as the main PHP executable or DLL your web server will use (e.g. php.exe, php4apache.dll).
  The PHP binary, the SAPI modules, and some extensions rely on external DLLs for execution. Make sure that these DLLs in the distribution exist in a directory that is in the Windows PATH. For example, if you enable php_oci8.dll in php.ini then you'll want to make sure the Oracle home directory can be seen in PATH so PHP can find oci.dll.
  The best bet to do it is to copy the files below into your system directory, which is typically:

  c:\windows\system for Windows 9x/ME

  c:\winnt\system32 for Windows NT/2000

  c:\windows\system32 for Windows XP

  The files to copy are:

  php4ts.dll, if it already exists there, overwrite it

  The files in your distribution's 'dlls' directory. If you have them already installed on your system, overwrite them only if something doesn't work correctly (Before overwriting them, it is a good idea to make a backup of them, or move them to another folder - just in case something goes wrong).

  Download the latest version of the Microsoft Data Access Components (MDAC) for your platform, especially if you use Microsoft Windows 9x/NT4. MDAC is available at http://www.microsoft.com/data/.
- Copy your chosen ini file (see below) to your '%WINDOWS%' directory on Windows 9x/Me or to your '%SYSTEMROOT%' directory under Windows NT/2000/XP and rename it to php.ini. Your '%WINDOWS%' or '%SYSTEMROOT%' directory is typically:

  c:\windows for Windows 9x/ME/XP

  c:\winnt or c:\winnt40 for NT/2000 servers

  There are two ini files distributed in the zip file, php.ini-dist and php.ini-optimized. We advise you to use php.ini-optimized, because we optimized the default settings in this file for performance, and security. The best is to study all the ini settings and set every element manually yourself. If you would like to achieve the best security, then this is the way for you, although PHP works fine with these default ini files.
- Edit your new php.ini file:
  - You will need to change the 'extension_dir' setting to point to your php-install-dir, or where you have placed your php_*.dll files. Please do not forget the last backslash. ex: c:\php\extensions\
  - If you are using OmniHTTPd, do not follow the next step. Set the 'doc_root' to point to your webservers document_root. For example: c:\apache\htdocs or c:\webroot

php_*.dll files. Please do not forget the last backslash. ex: c:\php\extensions\
    - If you are using OmniHTTPd, do not follow the next step. Set the 'doc_root' to point to your webservers document_root. For example: c:\apache\htdocs or c:\webroot
    - Choose which extensions you would like to load when PHP starts. See the section about Windows extensions, about how to set up one, and what is already built in. Note that on a new installation it is advisable to first get PHP working and tested without any extensions before enabling them in php.ini.
    - On PWS and IIS, you can set the browscap.ini to point to: c:\windows\system\inetsrv\browscap.ini on Windows 9x/Me, c:\winnt\system32\inetsrv\browscap.ini on NT/2000, and c:\windows\system32\inetsrv\browscap.ini on XP.
    - Note that the mibs directory supplied with the Windows distribution contains support files for SNMP. This directory should be moved to DRIVE:\usr\mibs (DRIVE being the drive where PHP is installed.)
    - If you're using NTFS on Windows NT, 2000 or XP, make sure that the user running the webserver has read permissions to your php.ini (e.g. make it readable by Everyone).
- For PWS give execution permission to the webroot:
    - Start PWS Web Manager
    - Edit Properties of the "Home"-Directory
    - Select the "execute"-Checkbox

Building from source

Before getting started, it is worthwhile answering the question: "Why is building on Windows so hard?" Two reasons come to mind:

1. Windows does not (yet) enjoy a large community of developers who are willing to freely share their source. As a direct result, the necessary investment in infrastructure required to support such development hasn't been made. By and large, what is available has been made possible by the porting of necessary utilities from Unix. Don't be surprised if some of this heritage shows through from time to time.
2. Pretty much all of the instructions that follow are of the "set and forget" variety. So sit back and try follow the instructions below as faithfully as you can.

Requirements

To compile and build PHP you need a Microsoft Development Environment. Microsoft Visuaul C++ 6.0 is recommended. To extract the downloaded files you need a extraction utilitiy (e.g.: Winzip). If you don't already have an unzip utility, you can get a free version from InfoZip.

Before you get started, you have to download...

- ..the win32 buildtools from the PHP site at http://www.php.net/extra/win32build.zip.
- ..the source code for the DNS name resolver used by PHP from http://www.php.net/extra/bindlib_w32.zip. This is a replacement for the resolv.lib library included in win32build.zip.
- If you plan to compile PHP as a Apache module you will also need the Apache sources.

Finally, you are going to need the source to PHP 4 itself. You can get the latest development version using anonymous CVS, a snapshot or the most recent released source tarball.

Putting it all together

After downloading the required packages you have to extract them in a proper place.

- Create a working directory where all files end up after extracting, e.g: c:\work.
- Create the directory win32build under your working directory (c:\work) and unzip win32build.zip into it.
- Create the directory bindlib_w32 under your working directory (c:\work) and unzip bindlib_w32.zip into it.
- Extract the downloaded PHP source code into your working directory (c:\work).

Following this steps your directory structure looks like this:

```
+--c:\work
| |
| +--bindlib_w32
| | |
| | +--arpa
| | |
| | +--conf
| | |
| | +--...
| |
| +--php-4.x.x
| | |
| | +--build
| | |
| | +--...
| | |
| | +--win32
| | |
| | +--...
| |
| +--win32build
| | |
| | +--bin
| | |
| | +--include
| | |
| | +--lib
```

Create the directories c:\usr\local\lib. Copy bison.simple from c:\work\win32build\bin to c:\usr\local\lib.

> **Note:** Cygwin users may omit the last step. A properly installed Cygwin environment provides the mandatory files bison.simple and bison.exe.

Configure MVC ++

The next step is to configure MVC ++ to prepare for compiling. Launch Microsoft Visual C++, and from the menu select Tools => Options. In the dialog, select the directories tab. Sequentially change the dropdown to Executables, Includes, and Library files. Your entries should look like this:

The next step is to configure MVC ++ to prepare for compiling. Launch Microsoft Visual C++, and from the menu select Tools -> Options. In the dialog, select the directories tab. Sequentially change the dropdown to Executables, Includes, and Library files. Your entries should look like this:

- Executable files: c:\work\win32build\bin, Cygwin users: cygwin\bin
- Include files: c:\work\win32build\include
- Library files: c:\work\win32build\lib

Build resolv.lib

You must build the resolv.lib library. Decide whether you want to have debug symbols available (bindlib - Win32 Debug) or not (bindlib - Win32 Release). Build the appropriate configuration:

- For GUI users, launch VC++, and then select File => Open Workspace, navigate to c:\work\bindlib_w32and select bindlib.dsw. Then select Build=>Set Active Configuration and select the desired configuration. Finally select Build=>Rebuild All.
- For command line users, make sure that you either have the C++ environment variables registered, or have run **vcvars.bat**, and then execute one of the following commands:
  - **msdev bindlib.dsp /MAKE "bindlib - Win32 Debug"**
  - **msdev bindlib.dsp /MAKE "bindlib - Win32 Release"**

At this point, you should have a usable resolv.lib in either your c:\work\bindlib_w32\Debug or Release subdirectories. Copy this file into your c:\work\win32build\lib directory over the file by the same name found in there.

Compiling

The best way to get started is to build the CGI version.

- For GUI users, launch VC++, and then select File => Open Workspace and select c:\work\php-4.x.x\win32\php4ts.dsw . Then select Build=>Set Active Configuration and select the desired configuration, either php4ts - Win32 Debug_TS or php4ts - Win32 Release_TS. Finally select Build=>Rebuild All.
- For command line users, make sure that you either have the C++ environment variables registered, or have run **vcvars.bat**, and then execute one of the following commands from the c:\work\php-4.x.x\win32 directory:
  - **msdev php4ts.dsp /MAKE "php4ts - Win32 Debug_TS"**
  - **msdev php4ts.dsp /MAKE "php4ts - Win32 Release_TS"**
  - At this point, you should have a usable php.exe in either your c:\work\php-4.x.x.\Debug_TS or Release_TS subdirectories.

It is possible to do minor customization to the build process by editing the main/config.win32.h.in file. For example you can change the builtin extensions, the location of php.ini and

Next you may want to build the CLI version which is designed to use PHP from the command line. The steps are the same as for building the CGI version, except you have to select the php4ts_cli - Win32 Debug_TS or php4ts_cli - Win32 Release_TS project file. After a succcessfull compiling run you will find the php.exe in either the directory Release_TS\cli\ or Debug_TS\cli\.

> **Note:** If you want to use PEAR and the comfortable command line installer, the CLI-SAPI is mandatory. For more information about PEAR and the installer read the documantation at the PEAR website.

In order to build the SAPI module (php4isapi.dll for integrating PHP with Microsoft IIS, set your active configuration to php4isapi-whatever-config and build the desired dll.

Installation of Windows extensions

After installing PHP and a webserver on Windows, you will probably want to install some extensions for added functionality. The following table describes some of the extensions available. You can choose which extensions you would like to load when PHP starts by uncommenting the: 'extension=php_*.dll' lines in php.ini. You can also load a module dynamically in your script using **dl()**.

The DLLs for PHP extensions are prefixed with 'php_' in PHP 4 (and 'php3_' in PHP 3). This prevents confusion between PHP extensions and their supporting libraries.

> **Note:** In PHP 4.0.6 BCMath, Calendar, COM, FTP, MySQL, ODBC, PCRE, Session, WDDX and XML support is built in. You don't need to load any additional extensions in order to use these functions. See your distributions README.txt or install.txt for a list of built in modules.
>
> **Note:** Some of these extensions need extra DLLs to work. Couple of them can be found in the distribution package, in the 'dlls' folder but some, for example Oracle (php_oci8.dll) require DLLs which are not bundled with the distribution package.
>
> Copy the bundled DLLs from 'DLLs' folder to your Windows PATH, safe places are:
>
> c:\windows\system for Windows 9x/Me
> c:\winnt\system32 for Windows NT/2000
> c:\windows\system32 for Windows XP
>
> If you have them already installed on your system, overwrite them only if something doesn't work correctly (Before overwriting them, it is a good idea to make a backup of them, or move them to another folder - just in case something goes wrong).

Table 3-1. PHP Extensions

Extension	Description	Notes
php_bz2.dll	bzip2 compression functions	None
php_calendar.dll	Calendar conversion functions	Built in since PHP 4.0.3
php_cpdf.dll	ClibPDF functions	None
php_crack.dll	Crack functions	None
php3_crypt.dll	Crypt functions	unknown
php_ctype.dll	ctype family functions	None
php_curl.dll	CURL, Client URL library functions	Requires: libeay32.dll, ssleay32.dll (bundled)
php_cybercash.dll	Cybercash payment functions	None

php_curl.dll	CURL, Client URL library functions	Requires: libeay32.dll, ssleay32.dll (bundled)
php_cybercash.dll	Cybercash payment functions	None
php_db.dll	DBM functions	Deprecated. Use DBA instead (php_dba.dll)
php_dba.dll	DBA: DataBase (dbm-style) Abstraction layer functions	None
php_dbase.dll	dBase functions	None
php3_dbm.dll	Berkeley DB2 library	unknown
php_domxml.dll	DOM XML functions	Requires: libxml2.dll (bundled)
php_dotnet.dll	.NET functions	None
php_exif.dll	Read EXIF headers from JPEG	None
php_fbsql.dll	FrontBase functions	None
php_fdf.dll	FDF: Forms Data Format functions.	Requires: fdftk.dll (bundled)
php_filepro.dll	filePro functions	Read-only access
php_ftp.dll	FTP functions	Built-in since PHP 4.0.3
php_gd.dll	GD library image functions	None
php_gd2.dll	GD2 library image functions	None
php_gettext.dll	Gettext functions	Requires: gnu_gettext.dll (bundled)
php_hyperwave.dll	HyperWave functions	None
php_iconv.dll	ICONV characterset conversion	Requires: iconv-1.3.dll (bundled)
php_ifx.dll	Informix functions	Requires: Informix libraries
php_iisfunc.dll	IIS management functions	None
php_imap.dll	IMAP POP3 and NNTP functions	PHP 3: php3_imap4r1.dll
php_ingres.dll	Ingres II functions	Requires: Ingres II libraries
php_interbase.dll	InterBase functions	Requires: gds32.dll (bundled)
php_java.dll	Java extension	Requires: jvm.dll (bundled)
php_ldap.dll	LDAP functions	Requires: libsasl.dll (bundled)
php_mhash.dll	Mhash Functions	None
php_ming.dll	Ming functions for Flash	None
php_msql.dll	mSQL functions	Requires: msql.dll (bundled)
php3_msql1.dll	mSQL 1 client	unknown
php3_msql2.dll	mSQL 2 client	unknown
php_mssql.dll	MSSQL functions	Requires: ntwdblib.dll (bundled)
php3_mysql.dll	MySQL functions	Built-in in PHP 4
php3_nsmail.dll	Netscape mail functions	unknown
php3_oci73.dll	Oracle functions	unknown
php_oci8.dll	Oracle 8 functions	Requires: Oracle 8 client libraries
php_openssl.dll	OpenSSL functions	Requires: libeay32.dll (bundled)
php_oracle.dll	Oracle functions	Requires: Oracle 7 client libraries
php_pdf.dll	PDF functions	None
php_pgsql.dll	PostgreSQL functions	None
php_printer.dll	Printer functions	None
php_xslt.dll	XSLT functions	Requires: sablot.dll (bundled)
php_snmp.dll	SNMP get and walk functions	NT only!
php_sybase_ct.dll	Sybase functions	Requires: Sybase client libraries
php_yaz.dll	YAZ functions	None
php_zlib.dll	ZLib compression functions	None

Servers-CGI/Commandline

The default is to build PHP as a CGI program. This creates a commandline interpreter, which can be used for CGI processing, or for non-web-related PHP scripting. If you are running a web server PHP has module support for, you should generally go for that solution for performance reasons. However, the CGI version enables Apache users to run different PHP-enabled pages under different user-ids. Please make sure you read through the Security chapter if you are going to run PHP as a CGI.

As of PHP 4.3.0, some important additions have happened to PHP. A new SAPI named CLI also exists and it has the same name as the CGI binary. What is installed at {PREFIX}/bin/php depends on your configure line and this is described in detail in the manual section named Using PHP from the command line. For further details please read that section of the manual.

Testing

If you have built PHP as a CGI program, you may test your build by typing **make test**. It is always a good idea to test your build. This way you may catch a problem with PHP on your platform early instead of having to struggle with it later.

Benchmarking

If you have built PHP 3 as a CGI program, you may benchmark your build by typing **make bench**. Note that if Safe Mode is on by default, the benchmark may not be able to finish if it takes longer then the 30 seconds allowed. This is because the **set_time_limit()** can not be used in safe mode. Use the max_execution_time configuration setting to control this time for your own scripts. **make bench** ignores the configuration file.

> **Note:** **make bench** is only available for PHP 3.

Using Variables

Some server supplied enviroment variables are not defined in the current CGI/1.1 specification. Only the following variables are defined there; everything else should be treated as 'vendor extensions': AUTH_TYPE, CONTENT_LENGTH, CONTENT_TYPE, GATEWAY_INTERFACE, PATH_INFO, PATH_TRANSLATED, QUERY_STRING, REMOTE_ADDR, REMOTE_HOST,

Some server supplied environment variables are not defined in the current CGI/1.1 specification. Only the following variables are defined there: everything else should be treated as 'vendor extensions': AUTH_TYPE, CONTENT_LENGTH, CONTENT_TYPE, GATEWAY_INTERFACE, PATH_INFO, PATH_TRANSLATED, QUERY_STRING, REMOTE_ADDR, REMOTE_HOST, REMOTE_IDENT, REMOTE_USER, REQUEST_METHOD, SCRIPT_NAME, SERVER_NAME, SERVER_PORT, SERVER_PROTOCOL and SERVER_SOFTWARE

Servers-Apache

This section contains notes and hints specific to Apache installs of PHP, both for Unix and Windows versions. We also have instructions and notes for Apache 2 on a separate page.

Details of installing PHP with Apache on Unix

You can select arguments to add to the **configure** on line 10 below from the Complete list of configure options. The version numbers have been omitted here, to ensure the instructions are not incorrect. You will need to replace the 'xxx' here with the correct values from your files.

Example 3-5. Installation Instructions (Apache Shared Module Version) for PHP 4

1. gunzip apache_xxx.tar.gz
2. tar -xvf apache_xxx.tar
3. gunzip php-xxx.tar.gz
4. tar -xvf php-xxx.tar
5. cd apache_xxx
6. ./configure --prefix=/www --enable-module=so
7. make
8. make install
9. cd ../php-xxx
10. ./configure --with-mysql --with-apxs=/www/bin/apxs
11. make
12. make install

    If you decide to change your configure options after installation
    you only need to repeat the last three steps. You only need to
    restart apache for the new module to take effect. A recompile of
    Apache is not needed.

13. cp php.ini-dist /usr/local/lib/php.ini

    You can edit your .ini file to set PHP options. If
    you prefer this file in another location, use
    --with-config-file-path=/path in step 10.

14. Edit your httpd.conf or srm.conf file and check that these lines are
    present and not commented out:

    AddType application/x-httpd-php .php

    LoadModule php4_module libexec/libphp4.so

    You can choose any extension you wish here. .php is simply the one
    we suggest. You can even include .html, and .php3 can be added for
    backwards compatibility.

    The path on the right hand side of the LoadModule statement must point
    to the path of the PHP module on your system. The above statement is
    correct for the steps shown above.

15. Use your normal procedure for starting the Apache server. (You must
    stop and restart the server, not just cause the server to reload by
    use a HUP or USR1 signal.)

Depending on your Apache install and Unix variant, there are many possible ways to stop and restart the server. Below are some typical lines used in restarting the server, for different apache/unix installations. You should replace /path/to/ with the path to these applications on your systems.

1. Several Linux and SysV variants:
/etc/rc.d/init.d/httpd restart

2. Using apachectl scripts:
/path/to/apachectl stop
/path/to/apachectl start

3. httpdctl and httpsdctl (Using OpenSSL), similar to apachectl:
/path/to/httpsdctl stop
/path/to/httpsdctl start

4. Using mod_ssl, or another SSL server, you may want to manually
stop and start:
/path/to/apachectl stop
/path/to/apachectl startssl

The locations of the apachectl and http(s)dctl binaries often vary. If your system has locate or whereis or which commands, these can assist you in finding your server control programs.

Different examples of compiling PHP for apache are as follows:

./configure --with-apxs --with-pgsql

```
./configure --with-apxs --with-pgsql
```

This will create a libphp4.so shared library that is loaded into Apache using a LoadModule line in Apache's httpd.conf file. The PostgreSQL support is embedded into this libphp4.so library.

```
./configure --with-apxs --with-pgsql=shared
```

This will create a libphp4.so shared library for Apache, but it will also create a pgsql.so shared library that is loaded into PHP either by using the extension directive in php.ini file or by loading it explicitly in a script using the **dl()** function.

```
./configure --with-apache=/path/to/apache_source --with-pgsql
```

This will create a libmodphp4.a library, a mod_php4.c and some accompanying files and copy this into the src/modules/php4 directory in the Apache source tree. Then you compile Apache using --activate-module=src/modules/php4/libphp4.a and the Apache build system will create libphp4.a and link it statically into the httpd binary. The PostgreSQL support is included directly into this httpd binary, so the final result here is a single httpd binary that includes all of Apache and all of PHP.

```
./configure --with-apache=/path/to/apache_source --with-pgsql=shared
```

Same as before, except instead of including PostgreSQL support directly into the final httpd you will get a pgsql.so shared library that you can load into PHP from either the php.ini file or directly using **dl()**.

When choosing to build PHP in different ways, you should consider the advantages and drawbacks of each method. Building as a shared object will mean that you can compile apache separately, and don't have to recompile everything as you add to, or change, PHP. Building PHP into apache (static method) means that PHP will load and run faster. For more information, see the Apache webpage on DSO support.

> **Note:** Apache's default http.conf currently ships with a section that looks like this:
>
> ```
> User nobody
> Group "#-1"
> ```
>
> Unless you change that to "Group nogroup" or something like that ("Group daemon" is also very common) PHP will not be able to open files.
>
> **Note:** Make sure you specify the installed version of apxs when using --with-apxs=/path/to/apxs. You must NOT use the apxs version that is in the apache sources but the one that is actually installed on your system.

Installing PHP on Windows with Apache 1.3.x

There are two ways to set up PHP to work with Apache 1.3.x on Windows. One is to use the CGI binary (php.exe), the other is to use the Apache module DLL. In either case you need to stop the Apache server, and edit your srm.conf or httpd.conf to configure Apache to work with PHP.

It is worth noting here that now the SAPI module has been made more stable under windows, we recommend it's use above the CGI binary, since it is more transparent and secure.

Although there can be a few variations of configuring PHP under Apache, these are simple enough to be used by the newcomer. Please consult the Apache Docs for further configuration directives.

If you unziped the PHP package to c:\php\ as described in the Manual Installation Steps section, you need to insert these lines to your Apache configuration file to set up the CGI binary:

- ScriptAlias /php/ "c:/php/"
- AddType application/x-httpd-php .php .phtml
- Action application/x-httpd-php "/php/php.exe"

Note that the second line in the list above can be found in the actual versions of httpd.conf, but it is commented out. Remember also to substitute the c:/php/ for your actual path to PHP.

Warning
By using the CGI setup, your server is open to several possible attacks. Please read our CGI security section to learn how to defend yourself from attacks.

If you would like to use PHP as a module in Apache, be sure to move php4ts.dll to the windows/system (for Windows 9x/Me) or winnt/system32 (for Windows NT/2000/XP) directory, overwriting any older file. Then you should add the following two lines to you Apache conf file:

- LoadModule php4_module c:/php/sapi/php4apache.dll
- AddType application/x-httpd-php .php .phtml

After changing the configuration file, remember to restart the server, for example, NET STOP APACHE followed by NET START APACHE, if you run Apache as a Windows Service, or use your regular shortcuts.

> **Note:** You may find after using the windows installer for Apache that you need to define the AddModule directive for mod_php4.c in the configuration file (httpd.conf). This is done by adding AddModule mod_php4.c to the AddModule list, near the beginning of the configuration file. This is especially important if the ClearModuleList directive is defined. Failure to do this may mean PHP will not be registered as an Apache module.

There are 2 ways you can use the source code highlighting feature, however their ability to work depends on your installation. If you have configured Apache to use PHP as an ISAPI module, then by adding the following line to your configuration file you can use this feature: AddType application/x-httpd-php-source .phps

If you chose to configure Apache to use PHP as a CGI binary, you will need to use the **show_source()** function. To do this simply

this feature: AddType application/x-httpd-php-source .phps

If you chose to configure Apache to use PHP as a CGI binary, you will need to use the **show_source()** function. To do this simply create a PHP script file and add this code: <?php show_source ("original_php_script.php"); ?>. Substitute original_php_script.php with the name of the file you wish to show the source of.

> **Note:** On Win-Apache all backslashes in a path statement such as "c:\directory\file.ext", must be converted to forward slashes, as "c:/directory/file.ext".

Servers-Apache 2.0

This section contains notes and hints specific to Apache 2.0 installs of PHP, both for Unix and Windows versions.

Warning
Do not use Apache 2.0 and PHP in a production environment neither on Unix nor on Windows.

You are highly encouraged to take a look at the Apache Documentation to get a basic understanding of the Apache 2.0 Server.

PHP and Apache 2.0 compatibility notes

The following versions of PHP are known to work with the most recent version of Apache 2.0:

- PHP 4.3.0 or later available at http://www.php.net/downloads.php.
- the latest stable development version. Get the source code http://snaps.php.net/php4-latest.tar.gz or download binaries for windows http://snaps.php.net/win32/php4-win32-latest.zip.
- a prerelease version downloadable from http://qa.php.net/.
- you have always the option to obtain PHP through anonymous CVS.

These versions of PHP are compatible to Apache 2.0.40 and later.

> **Note:** Apache 2.0 SAPI-support started with PHP 4.2.0. PHP 4.2.3 its known to work in conjunction with Apache 2.0.39. Don't try to use this version of PHP with any other version of Apache. We do not recommend to use PHP 4.2.3 along with Apache 2.0.39.
>
> All mentioned versions of PHP will work still with Apache 1.3.x.

PHP and Apache 2 on Linux

Download the most recent version of Apache 2.0 and a fitting PHP version from the above mentioned places. This quick guide covers only the basics to get started with Apache 2.0 and PHP. For more information read the Apache Documentation. The version numbers have been omitted here, to ensure the instructions are not incorrect. You will need to replace the 'NN' here with the correct values from your files.

Example 3-6. Installation Instructions (Apache 2 Shared Module Version)

1. gzip -d httpd-2_0_NN.tar.gz
2. tar xvf httpd-2_0_NN.tar
3. gunzip php-NN.tar.gz
4. tar -xvf php-NN.tar
5. cd httpd-2_0_NN
6. ./configure --enable-so
7. make
8. make install

   Now you have Apache 2.0.NN available under /usr/local/apache2,
   configured with loadable module support and the standard MPM prefork.
   To test the installation use your normal procedure for starting
   the Apache server, e.g.:
   /usr/local/apache2/bin/apachectl start
   and stop the server to go on with the configuration for PHP:
   /usr/local/apache2/bin/apachectl stop.

9. cd ../php4-NN
10. ./configure --with-apxs2=/usr/local/apache2/bin/apxs
11. make
12. make install
13. cp php.ini-dist /usr/local/lib/php.ini

    Edit your php.ini file to set PHP options. If
    you prefer this file in another location, use
    --with-config-file-path=/path in step 10.

14. Edit your httpd.conf file and check that these lines are
    present:

    LoadModule php4_module modules/libphp4.so
    AddType application/x-httpd-php .php

    You can choose any extension you wish here. .php is simply the one
    we suggest.

    The path on the right hand side of the LoadModule statement must point
    to the path of the PHP module on your system. The above statement is
    correct for the steps shown above.

15. Use your normal procedure for starting the Apache server, e.g.:
    /usr/local/apache2/bin/apachectl start

Following the steps above you will have a running Apache 2.0 with support for PHP as SAPI module. Of course there are many more configuration options available for both, Apache and PHP. For more information use **./configure --help** in the corresponding source tree. In case you wish to build a multithreaded version of Apache 2.0 you must overwrite the standard MPM-Module prefork either

Following the steps above you will have a running Apache 2.0 with support for PHP as SAPI module. Of course there are many more configuration options available for both, Apache and PHP. For more information use **./configure --help** in the corresponding source tree. In case you wish to build a multithreaded version of Apache 2.0 you must overwrite the standard MPM-Module prefork either with worker or perchild. To do so append to your configure line in step 6 above either the option --with-mpm=worker or --with-mpm=perchild. Take care about the consequences and understand what you are doing. For more information read the Apache documentation about the MPM-Modules.

> **Note:** To build a multithreaded version of Apache your system must support threads. This also implies to build PHP with experimental Zend Thread Safety (ZTS). Therefore not all extensions might be available. The recommended setup is to build Apache with the standard prefork MPM-Module.

PHP and Apache 2.0 on Windows

Consider to read the Windows specific notes for Apache 2.0.

Warning
Apache 2.0 is designed to run on Windows NT 4.0, Windows 2000 or Windows XP. At this time, support for Windows 9x is incomplete. Apache 2.0 is not expected to work on those platforms at this time.

Download the most recent version of Apache 2.0 and a fitting PHP version from the above mentioned places. Follow the Manual Installation Steps and come back to go on with the integration of PHP and Apache.

There are two ways to set up PHP to work with Apache 2.0 on Windows. One is to use the CGI binary the other is to use the Apache module DLL. In either case you need to stop the Apache server, and edit your httpd.conf to configure Apache to work with PHP.

You need to insert these three lines to your Apache httpd.conf configuration file to set up the CGI binary:

Example 3-7. PHP and Apache 2.0 as CGI

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"
```

If you would like to use PHP as a module in Apache 2.0, be sure to move php4ts.dll to winnt/system32 (for Windows NT/2000) or windows/system32 (for Windows XP), overwriting any older file. You need to insert these two lines to your Apache httpd.conf configuration file to set up the PHP-Module for Apache 2.0:

Example 3-8. PHP and Apache 2.0 as Module

```
LoadModule php4_module c:/php/sapi/php4apache2.dll
AddType application/x-httpd-php .php
```

> **Note:** Remember to substitute the c:/php/ for your actual path to PHP in the above examples.

Warning
Don't mix up your installation with dll files from different PHP versions . You have the only choice to use the dll's and extensions that ship with your downloaded PHP version.

Servers-Caudium

PHP 4 can be built as a Pike module for the Caudium webserver. Note that this is not supported with PHP 3. Follow the simple instructions below to install PHP 4 for Caudium.

Example 3-9. Caudium Installation Instructions

1. Make sure you have Caudium installed prior to attempting to install PHP 4. For PHP 4 to work correctly, you will need Pike 7.0.268 or newer. For the sake of this example we assume that Caudium is installed in /opt/caudium/server/.
2. Change directory to php-x.y.z (where x.y.z is the version number).
3. ./configure --with-caudium=/opt/caudium/server
4. make
5. make install
6. Restart Caudium if it's currently running.
7. Log into the graphical configuration interface and go to the virtual server where you want to add PHP 4 support.
8. Click Add Module and locate and then add the PHP 4 Script Support module.
9. If the documentation says that the 'PHP 4 interpreter isn't available', make sure that you restarted the server. If you did check /opt/caudium/logs/debug/default.1 for any errors related to <filename>PHP4.so</filename>. Also make sure that <filename>caudium/server/lib/[pike-version]/PHP4.so</filename> is present.
10. Configure the PHP Script Support module if needed.

You can of course compile your Caudium module with support for the various extensions available in PHP 4. See the complete list of configure options for an exhaustive rundown.

> **Note:** When compiling PHP 4 with MySQL support you must make sure that the normal MySQL client code is used. Otherwise there might be conflicts if your Pike already has MySQL support. You do this by specifying a MySQL install directory the --with-mysql option.

Servers-fhttpd

To build PHP as an fhttpd module, answer "yes" to "Build as an fhttpd module?" (the --with-fhttpd=DIR option to configure) and specify the fhttpd source base directory. The default directory is /usr/local/src/fhttpd. If you are running fhttpd, building PHP as a module will give better performance, more control and remote execution capability.

---

specify the thttpd source base directory. The default directory is /usr/local/src/thttpd. If you are running thttpd, building PHP as a module will give better performance, more control and remote execution capability.

Servers-IIS/PWS

This section contains notes and hints specific to IIS (Microsoft Internet Information Server). Installing PHP for PWS/IIS 3, PWS 4 or newer and IIS 4 or newer versions.

> **Important for CGI users:** Read the faq on cgi.force_redirect for important details. This directive needs to be set to 0.

Windows and PWS/IIS 3

The recommended method for configuring these servers is to use the REG file included with the distribution (pws-php4cgi.reg). You may want to edit this file and make sure the extensions and PHP install directories match your configuration. Or you can follow the steps below to do it manually.

Warning
These steps involve working directly with the Windows registry. One error here can leave your system in an unstable state. We highly recommend that you back up your registry first. The PHP Development team will not be held responsible if you damage your registry.

- Run Regedit.
- Navigate to: HKEY_LOCAL_MACHINE /System /CurrentControlSet /Services /W3Svc /Parameters /ScriptMap.
- On the edit menu select: New->String Value.
- Type in the extension you wish to use for your php scripts. For example .php
- Double click on the new string value and enter the path to php.exe in the value data field. ex: c:\php\php.exe.
- Repeat these steps for each extension you wish to associate with PHP scripts.

The following steps do not affect the web server installation and only apply if you want your php scripts to be executed when they are run from the command line (ex. run c:\myscripts\test.php) or by double clicking on them in a directory viewer window. You may wish to skip these steps as you might prefer the PHP files to load into a text editor when you double click on them.

- Navigate to: HKEY_CLASSES_ROOT
- On the edit menu select: New->Key.
- Name the key to the extension you setup in the previous section. ex: .php
- Highlight the new key and in the right side pane, double click the "default value" and enter phpfile.
- Repeat the last step for each extension you set up in the previous section.
- Now create another New->Key under HKEY_CLASSES_ROOT and name it phpfile.
- Highlight the new key phpfile and in the right side pane, double click the "default value" and enter PHP Script.
- Right click on the phpfile key and select New->Key, name it Shell.
- Right click on the Shell key and select New->Key, name it open.
- Right click on the open key and select New->Key, name it command.
- Highlight the new key command and in the right side pane, double click the "default value" and enter the path to php.exe. ex: c:\php\php.exe -q %1. (don't forget the %1).
- Exit Regedit.
- If using PWS on Windows, reboot to reload the registry.

PWS and IIS 3 users now have a fully operational system. IIS 3 users can use a nifty tool from Steven Genusa to configure their script maps.

Windows and PWS 4 or newer

When installing PHP on Windows with PWS 4 or newer version, you have two options. One to set up the PHP CGI binary, the other is to use the ISAPI module DLL.

If you choose the CGI binary, do the following:

- Edit the enclosed pws-php4cgi.reg file (look into the SAPI dir) to reflect the location of your php.exe. Backslashes should be escaped, for example: [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]".php"="c:\\php\\php.exe" Now merge this registery file into your system; you may do this by double-clicking it.
- In the PWS Manager, right click on a given directory you want to add PHP support to, and select Properties. Check the 'Execute' checkbox, and confirm.

If you choose the ISAPI module, do the following:

- Edit the enclosed pws-php4isapi.reg file (look into the SAPI dir) to reflect the location of your php4isapi.dll. Backslashes should be escaped, for example:
  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
  ".php"="c:\\php\\sapi\\php4isapi.dll" Now merge this registery file into your system; you may do this by double-clicking it.
- In the PWS Manager, right click on a given directory you want to add PHP support to, and select Properties. Check the 'Execute' checkbox, and confirm.

Windows NT/2000/XP and IIS 4 or newer

To install PHP on an NT/2000/XP Server running IIS 4 or newer, follow these instructions. You have two options to set up PHP, using the CGI binary (php.exe) or with the ISAPI module.

In either case, you need to start the Microsoft Management Console (may appear as 'Internet Services Manager', either in your Windows NT 4.0 Option Pack branch or the Control Panel=>Administrative Tools under Windows 2000/XP). Then right click on your Web server node (this will most probably appear as 'Default Web Server'), and select 'Properties'.

If you want to use the CGI binary, do the following:

- Under 'Home Directory', 'Virtual Directory', or 'Directory', click on the 'Configuration' button, and then enter the App Mappings tab.
- Click Add, and in the Executable box, type: c:\php\php.exe (assuming that you have unziped PHP in c:\php\).
- In the Extension box, type the file name extension you want associated with PHP scripts. Leave 'Method exclusions' blank, and check the Script engine checkbox. You may also like to check the 'check that file exists' box - for a small performance

- In the Extension box, type the file name extension you want associated with PHP scripts. Leave 'Method exclusions' blank, and check the Script engine checkbox. You may also like to check the 'check that file exists' box - for a small performance penalty, IIS (or PWS) will check that the script file exists and sort out authentication before firing up php. This means that you will get sensible 404 style error messages instead of cgi errors complaining that php did not output any data. You must start over from the previous step for each extension you want associated with PHP scripts. .php and .phtml are common, although .php3 may be required for legacy applications.
- Set up the appropriate security. (This is done in Internet Service Manager), and if your NT Server uses NTFS file system, add execute rights for I_USR_ to the directory that contains php.exe.

To use the ISAPI module, do the following:

- If you don't want to perform HTTP Authentication using PHP, you can (and should) skip this step. Under ISAPI Filters, add a new ISAPI filter. Use PHP as the filter name, and supply a path to the php4isapi.dll.
- Under 'Home Directory', click on the 'Configuration' button. Add a new entry to the Application Mappings. Use the path to the php4isapi.dll as the Executable, supply .php as the extension, leave Method exclusions blank, and check the Script engine checkbox.
- Stop IIS completely (NET STOP iisadmin)
- Start IIS again (NET START w3svc)

Servers-Netscape and iPlanet

This section contains notes and hints specific to Netscape and iPlanet installs of PHP, both for Sun Solaris and Windows versions.

You can find more information about setting up PHP for the Netscape Enterprise Server here:
http://benoit.noss.free.fr/php/install-php4.html

Installing PHP with Netscape on Sun Solaris

To build PHP with NES or iPlanet web servers, enter the proper install directory for the --with-nsapi = DIR option. The default directory is usually /opt/netscape/suitespot/. Please also read /php-xxx-version/sapi/nsapi/nsapi-readme.txt.

Example 3-10. Installation Example for Netscape Enterprise on Solaris

```
Instructions for Sun Solaris 2.6 with Netscape Enterprise Server 3.6
From: bhager@invacare.com

1. Install the following packages from www.sunfreeware.com or another
download site:

   flex-2_5_4a-sol26-sparc-local
   gcc-2_95_2-sol26-sparc-local
   gzip-1.2.4-sol26-sparc-local
   perl-5_005_03-sol26-sparc-local
   bison-1_25-sol26-sparc-local
   make-3_76_1-sol26-sparc-local
   m4-1_4-sol26-sparc-local
   autoconf-2.13
   automake-1.4
   mysql-3.23.24-beta (if you want mysql support)
   tar-1.13 (GNU tar)

2. Make sure your path includes the proper directories
   PATH=.:/usr/local/bin:/usr/sbin:/usr/bin:/usr/ccs/bin
   export PATH

3. gunzip php-x.x.x.tar.gz (if you have a .gz dist, otherwise go to 4)
4. tar xvf php-x.x.x.tar
5. cd ../php-x.x.x

6. For the following step, make sure /opt/netscape/suitespot/ is where
  your netscape server is installed. Otherwise, change to correct path:
   /configure --with-mysql=/usr/local/mysql --with-nsapi=/opt/netscape/suitespot/ --enable-track-vars --enable-libgcc
7. make
8. make install
```

After performing the base install and reading the appropriate readme file, you may need to performs some additional configuration steps.

Firstly you may need to add some paths to the LD_LIBRARY_PATH environment for Netscape to find all the shared libs. This can best done in the start script for your Netscape server. Windows users can probably skip this step. The start script is often located in: /path/to/server/https-servername/start

You may also need to edit the configuration files that are located in:/path/to/server/https-servername/config/.

Example 3-11. Configuration Example for Netscape Enterprise

```
Configuration Instructions for Netscape Enterprise Server
From: bhager@invacare.com

1. Add the following line to mime.types:
   type=magnus-internal/x-httpd-php exts=php

2. Add the following to obj.conf, shlib will vary depending on
   your OS, for unix it will be something like
   /opt/netscape/suitespot/bin/libphp4.so.

   You should place the following lines after mime types init.
   Init fn="load-modules" funcs="php4_init,php4_close,php4_execute,php4_auth_trans" shlib="/php4/nsapiPHP4.dll"
   Init fn=php4_init errorString="Failed to initialize PHP!"

   <object name="default">
   .
```

<object name="default">
.
.
.
.#NOTE this next line should happen after all 'ObjectType' and before all 'AddLog' lines
Service fn="php4_execute" type="magnus-internal/x-httpd-php"
.
.
</Object>

<Object name="x-httpd-php">
ObjectType fn="force-type" type="magnus-internal/x-httpd-php"
Service fn=php4_execute
</Object>

Authentication configuration

PHP authentication cannot be used with any other authentication. ALL AUTHENTICATION IS PASSED TO YOUR PHP SCRIPT. To configure PHP Authentication for the entire server, add the following line:

<Object name="default">
AuthTrans fn=php4_auth_trans
.
.
.
.
</Object>

To use PHP Authentication on a single directory, add the following:

<Object ppath="d:\path\to\authenticated\dir*">
AuthTrans fn=php4_auth_trans
</Object>

If you are running Netscape Enterprise 4.x, then you should use the following:

Example 3-12. Configuration Example for Netscape Enterprise 4.x

```
Place these lines after the mime types init, and everything else is similar
to the example configuration above.
From: Graeme Hoose (GraemeHoose@BrightStation.com)

Init fn="load-modules" shlib="/path/to/server4/bin/libphp4.so" funcs="php4_init,php4_close,php4_execute,php4_auth_trans"
Init fn="php4_init" LateInit="yes"
```

Installing PHP with Netscape on Windows

To Install PHP as CGI (for Netscape Enterprise Server, iPlanet, perhaps Fastrack), do the following:

- Copy php4ts.dll to your systemroot (the directory where you installed windows)
- Make a file association from the command line. Type the following two lines:

  ```
  assoc .php=PHPScript
  ftype PHPScript=c:\php\php.exe %1 %*
  ```

- In the Netscape Enterprise Administration Server create a dummy shellcgi directory and remove it just after (this step creates 5 important lines in obj.conf and allow the web server to handle shellcgi scripts).
- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/shellcgi, File Suffix:php).
- Do it for each web server instance you want php to run

More details about setting up PHP as a CGI executable can be found here: http://benoit.noss.free.fr/php/install-php.html

To Install PHP as NSAPI (for Netscape Enterprise Server, iPlanet, perhaps Fastrack, do the following:

- Copy php4ts.dll to your systemroot (the directory where you installed windows)
- Make a file association from the command line. Type the following two lines:

  ```
  assoc .php=PHPScript
  ftype PHPScript=c:\php\php.exe %1 %*
  ```

- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/x-httpd-php, File Suffix:php).
- Stop your web service and edit obj.conf. At the end of the Init section, place these two lines (necessarily after mime type init!):

  ```
  Init fn="load-modules" funcs="php4_init,php4_close,php4_execute,php4_auth_trans" shlib="c:/php/sapi/php4nsapi.dll"
  Init fn="php4_init" errorString="Failed to initialise PHP!"
  ```

- In The < Object name="default" > section, place this line necessarily after all 'ObjectType' and before all 'AddLog' lines:

  ```
  Service fn="php4_execute" type="magnus-internal/x-httpd-php"
  ```

- At the end of the file, create a new object called x-httpd-php, by inserting these lines:

  ```
  <Object name="x-httpd-php">
  ObjectType fn="force-type" type="magnus-internal/x-httpd-php"
  Service fn=php4_execute
  </Object>
  ```

- Restart your web service and apply changes
- Do it for each web server instance you want PHP to run

More details about setting up PHP as an NSAPI filter can be found here: http://benoit.noss.free.fr/php/install-php4.html

---

Servers-OmniHTTPd Server

Servers-OmniHTTPd Server

This section contains notes and hints specific to OmniHTTPd.

OmniHTTPd 2.0b1 and up for Windows

You need to complete the following steps to make PHP work with OmniHTTPd. This is a CGI executable setup. SAPI is supported by OmniHTTPd, but some tests have shown that it is not so stable to use PHP as an ISAPI module.

> **Important for CGI users:** Read the faq on cgi.force_redirect for important details. This directive needs to be set to 0.

- Step 1: Install OmniHTTPd server.
- Step 2: Right click on the blue OmniHTTPd icon in the system tray and select Properties
- Step 3: Click on Web Server Global Settings
- Step 4: On the 'External' tab, enter: virtual = .php | actual = c:\path-to-php-dir\php.exe, and use the Add button.
- Step 5: On the Mime tab, enter: virtual = wwwserver/stdcgi | actual = .php, and use the Add button.
- Step 6: Click OK

Repeat steps 2 - 6 for each extension you want to associate with PHP.

> **Note:** Some OmniHTTPd packages come with built in PHP support. You can choose at setup time to do a custom setup, and uncheck the PHP component. We recommend you to use the latest PHP binaries. Some OmniHTTPd servers come with PHP 4 beta distributions, so you should choose not to set up the built in support, but install your own. If the server is already on your machine, use the Replace button in Step 4 and 5 to set the new, correct information.

Servers-Oreilly Website Pro

This section contains notes and hints specific to Oreilly Website Pro.

Oreilly Website Pro 2.5 and up for Windows

This list describes how to set up the PHP CGI binary or the ISAPI module to work with Oreilly Website Pro on Windows.

- Edit the Server Properties and select the tab "Mapping".
- From the List select "Associations" and enter the desired extension (.php) and the path to the CGI exe (ex. c:\php\php.exe) or the ISAPI DLL file (ex. c:\php\sapi\php4isapi.dll).
- Select "Content Types" add the same extension (.php) and enter the content type. If you choose the CGI executable file, enter 'wwwserver/shellcgi', if you choose the ISAPI module, enter 'wwwserver/isapi' (both without quotes).

Servers-Sambar

This section contains notes and hints specific to the Sambar server for Windows.

Sambar Windows

This list describes how to set up the ISAPI module to work with the Sambar server on Windows.

- Find the file called mappings.ini (in the config directory) in the Sambar isntall directory.
- Open mappings.ini and add the following line under [ISAPI]:

  ```
  *.php = c:\php\php4isapi.dll
  ```

  (This line assumes that PHP was installed in c:\php.)
- Now restart the Sambar server for the changes to take effect.

Servers-Xitami

This section contains notes and hints specific to Xitami.

Xitami for Windows

This list describes how to set up the PHP CGI binary to work with Xitami on Windows.

> **Important for CGI users:** Read the faq on cgi.force_redirect for important details. This directive needs to be set to 0.

- Make sure the webserver is running, and point your browser to xitamis admin console (usually http://127.0.0.1/admin), and click on Configuration.
- Navigate to the Filters, and put the extension which PHP should parse (i.e. .php) into the field File extensions (.xxx).
- In Filter command or script put the path and name of your php executable i.e. c:\php\php.exe.
- Press the 'Save' icon.
- Restart the server to reflect changes.

Servers-Other web servers

PHP can be built to support a large number of web servers. Please see Server-related options for a full list of server-related configure options. The PHP CGI binaries are compatible with almost all webservers supporting the CGI standard.

---

configure options. The PHP CGI binaries are compatible with almost all webservers supporting the CGI standard.

Problems?

Read the FAQ

Some problems are more common than others. The most common ones are listed in the PHP FAQ, part of this manual.

Other problems

If you are still stuck, someone on the PHP installation mailing list may be able to help you. You should check out the archive first, in case someone already answered someone else who had the same problem as you. The archives are available from the support page on http://www.php.net/. To subscribe to the PHP installation mailing list, send an empty mail to php-install-subscribe@lists.php.net. The mailing list address is php-install@lists.php.net.

If you want to get help on the mailing list, please try to be precise and give the necessary details about your environment (which operating system, what PHP version, what web server, if you are running PHP as CGI or a server module, etc.), and preferably enough code to make others able to reproduce and test your problem.

Bug reports

If you think you have found a bug in PHP, please report it. The PHP developers probably don't know about it, and unless you report it, chances are it won't be fixed. You can report bugs using the bug-tracking system at http://bugs.php.net/. Please do not send bug reports in mailing list or personal letters. The bug system is also suitable to submit feature requests.

Read the How to report a bug document before submitting any bug reports!

Complete list of configure options

> **Note:** These are only used at compile time. If you want to alter PHP's runtime configuration, please see the chapter on Configuration.

The following is a complete list of options supported by PHP 4 configure scripts (as of 4.1.0), used when compiling in Unix-like environments. Some are available in PHP 3, some in PHP 4, and some in both. This is not noted yet.

There are general configuration options for the **configure** script, consult the appropriate manual pages for GNU **autoconf** or use the command **configure --help** for a full, up-to-date list.

- Database
- Graphics
- Miscellaneous
- PHP Behaviour
- Server

Configure Options in PHP 4

> **Note:** These options are only used in PHP 4 as of PHP 4.1.0. Some are available in older versions of PHP 4, some even in PHP 3, some only in PHP 4.1.0. If you want to compile an older version, some options will probably not be available.

Database options

--with-dbplus

Include dbplus support.

--with-adabas[=DIR]

Include Adabas D support. DIR is the Adabas base install directory, defaults to /usr/local.

--with-sapdb[=DIR]

Include SAP DB support. DIR is SAP DB base install directory, defaults to /usr/local.

--with-solid[=DIR]

Include Solid support. DIR is the Solid base install directory, defaults to /usr/local/solid.

--with-ibm-db2[=DIR]

Include IBM DB2 support. DIR is the DB2 base install directory, defaults to /home/db2inst1/sqllib.

--with-empress[=DIR]

Include Empress support. DIR is the Empress base install directory, defaults to $EMPRESSPATH. From PHP4, this option only supports Empress Version 8.60 and above.

--with-empress-bcs[=DIR]

Include Empress Local Access support. DIR is the Empress base install directory, defaults to $EMPRESSPATH. From PHP4, this option only supports Empress Version 8.60 and above.

--with-birdstep[=DIR]

Include Birdstep support. DIR is the Birdstep base install directory, defaults to /usr/local/birdstep.

--with-custom-odbc[=DIR]

Include a user defined ODBC support. The DIR is ODBC install base directory, which defaults to /usr/local. Make sure to define CUSTOM_ODBC_LIBS and have some odbc.h in your include dirs. E.g., you should define following for Sybase SQL Anywhere 5.5.00 on QNX, prior to run configure script: CPPFLAGS="-DODBC_QNX -DSQLANY_BUG" LDFLAGS=-lunix CUSTOM_ODBC_LIBS="-ldblib -lodbc".

--with-iodbc[=DIR]

Include iODBC support. DIR is the iODBC base install directory, defaults to /usr/local.

--with-esoob[=DIR]

Include Easysoft OOB support. DIR is the OOB base install directory, defaults to /usr/local/easysoft/oob/client.

--with-unixODBC[=DIR]

Include unixODBC support. DIR is the unixODBC base install directory, defaults to /usr/local.

--with-unixODBC[=DIR]

Include unixODBC support. DIR is the unixODBC base install directory, defaults to /usr/local.

--with-openlink[=DIR]

Include OpenLink ODBC support. DIR is the OpenLink base install directory, defaults to /usr/local. This is the same as iODBC.

--with-dbmaker[=DIR]

Include DBMaker support. DIR is the DBMaker base install directory, defaults to where the latest version of DBMaker is installed (such as /home/dbmaker/3.6).

--disable-unified-odbc

Disable unified ODBC support. Only applicable if iODBC, Adabas, Solid, Velocis or a custom ODBC interface is enabled. PHP 3 only!

Graphics options

--without-gd

Disable GD support. PHP 3 only!

--with-imagick

The imagick extension has been moved to PECL in PEAR and can be found here. Install instructions for PHP 4 can be found on the PEAR site.

Simply doing --with-imagick is only supported in PHP 3 unless you follow the instructions found on the PEAR site.

--with-ming[=DIR]

Include ming support.

Misc options

--enable-force-cgi-redirect

Enable the security check for internal server redirects. You should use this if you are running the CGI version with Apache.

--enable-discard-path

If this is enabled, the PHP CGI binary can safely be placed outside of the web tree and people will not be able to circumvent .htaccess security.

--with-fastcgi=SRCDIR

Build PHP as FastCGI application.

--enable-debug

Compile with debugging symbols.

--with-layout=TYPE

Sets how installed files will be laid out. Type is one of PHP (default) or GNU.

--with-pear=DIR

Install PEAR in DIR (default PREFIX/lib/php).

--without-pear

Do not install PEAR.

--enable-sigchild

Enable PHP's own SIGCHLD handler.

--disable-rpath

Disable passing additional runtime library search paths.

--enable-libgcc

Enable explicitly linking against libgcc.

--enable-php-streams

Include experimental php streams. Do not use unless you are testing the code!

--with-zlib-dir=<DIR>

Define the location of zlib install directory.

--with-aspell[=DIR]

Include ASPELL support.

--with-ccvs[=DIR]

Include CCVS support.

--with-cybercash[=DIR]

Include CyberCash support. DIR is the CyberCash MCK install directory.

--with-icap[=DIR]

Include ICAP support.

--with-ircg-config

Path to the ircg-config script.

--with-ircg

Include ircg support.

--enable-mailparse

Enable mailparse support.

--with-muscat[=DIR]

Include muscat support.

--with-satellite[=DIR]

Enable CORBA support via Satellite (EXPERIMENTAL) DIR is the base directory for ORBit.

--enable-trans-sid

Enable transparent session id propagation.

--with-system-regex

Enable transparent session id propagation.
--with-system-regex

Use system regex library (deprecated).
--with-vpopmail[=DIR]

Include vpopmail support.
--with-tsrm-pthreads

Use POSIX threads (default).
--enable-shared[=PKGS]

Build shared libraries [default=yes].
--enable-static[=PKGS]

Build static libraries [default=yes].
--enable-fast-install[=PKGS]

Optimize for fast installation [default=yes].
--with-gnu-ld

Assume the C compiler uses GNU ld [default=no].
--disable-libtool-lock

Avoid locking (might break parallel builds).
--with-pic

Try to use only PIC/non-PIC objects [default=use both].
--enable-memory-limit

Compile with memory limit support.
--disable-url-fopen-wrapper

Disable the URL-aware fopen wrapper that allows accessing files via HTTP or FTP.
--enable-versioning

Export only required symbols. See INSTALL for more information.
--with-imsp[=DIR]

Include IMSp support (DIR is IMSP's include dir and libimsp.a dir). PHP 3 only!
--with-mck[=DIR]

Include Cybercash MCK support. DIR is the cybercash mck build directory, defaults to /usr/src/mck-3.2.0.3-linux for help look in extra/cyberlib. PHP 3 only!
--with-mod-dav=DIR

Include DAV support through Apache's mod_dav, DIR is mod_dav's installation directory (Apache module version only!) PHP 3 only!
--enable-debugger

Compile with remote debugging functions. PHP 3 only!
--enable-versioning

Take advantage of versioning and scoping provided by Solaris 2.x and Linux. PHP 3 only!

PHP options

--enable-maintainer-mode

Enable make rules and dependencies not useful (and sometimes confusing) to the casual installer.
--with-config-file-path=PATH

Sets the path in which to look for php.ini, defaults to PREFIX/lib.
--enable-safe-mode

Enable safe mode by default.
--with-exec-dir[=DIR]

Only allow executables in DIR when in safe mode defaults to /usr/local/php/bin.
--enable-magic-quotes

Enable magic quotes by default.
--disable-short-tags

Disable the short-form <? start tag by default.

Server options

--with-aolserver=DIR

Specify path to the installed AOLserver.
--with-apxs[=FILE]

Build shared Apache module. FILE is the optional pathname to the Apache apxs tool; defaults to apxs. Make sure you specify the version of apxs that is actually installed on your system and NOT the one that is in the apache source tarball.
--with-apache[=DIR]

Build Apache module. DIR is the top-level Apache build directory, defaults to /usr/local/apache.
--with-mod_charset

Enable transfer tables for mod_charset (Rus Apache).
--with-apxs2[=FILE]

Build shared Apache 2.0 module. FILE is the optional pathname to the Apache apxs tool; defaults to apxs.
--with-fhttpd[=DIR]

Build fhttpd module. DIR is the fhttpd sources directory, defaults to /usr/local/src/fhttpd.
--with-isapi=DIR

Build PHP as an ISAPI module for use with Zeus.
--with-nsapi=DIR

Specify path to the installed Netscape Server.

Specify path to the installed Netscape Server.

--with-phttpd=DIR

No information yet.

--with-pi3web=DIR

Build PHP as a module for use with Pi3Web.

--with-roxen=DIR

Build PHP as a Pike module. DIR is the base Roxen directory, normally /usr/local/roxen/server.

--enable-roxen-zts

Build the Roxen module using Zend Thread Safety.

--with-servlet[=DIR]

Include servlet support. DIR is the base install directory for the JSDK. This SAPI prereqs the java extension must be built as a shared dl.

--with-thttpd=SRCDIR

Build PHP as thttpd module.

--with-tux=MODULEDIR

Build PHP as a TUX module (Linux only).

Chapter 4. Configuration

The configuration file

The configuration file (called php3.ini in PHP 3.0, and simply php.ini as of PHP 4.0) is read when PHP starts up. For the server module versions of PHP, this happens only once when the web server is started. For the CGI and CLI version, it happens on every invocation.

The default location of php.ini is a compile time option (see the FAQ entry), but can be changed for the CGI and CLI version with the -c command line switch, see the chapter about using PHP from the command line. You can also use the environment variable PHPRC for an additional path to search for a php.ini file.

Not every PHP directive is documented below. For a list of all directives, please read your well commented php.ini file. You may want to view the latest php.ini here from CVS.

Note: The default value for the PHP directive register_globals changed from on to off in PHP 4.2.0.

Example 4-1. php.ini example

```
; any text on a line after an unquoted semicolon (;) is ignored
[php] ; section markers (text within square brackets) are also ignored
; Boolean values can be set to either:
;    true, on, yes
; or false, off, no, none
register_globals = off
magic_quotes_gpc = yes

; you can enclose strings in double-quotes
include_path = ".:/usr/local/lib/php"

; backslashes are treated the same as any other character
include_path = ".;c:\php\lib"
```

How to change configuration settings

Running PHP as Apache module

When using PHP as an Apache module, you can also change the configuration settings using directives in Apache configuration files (e.g. httpd.conf) and .htaccess files (You will need "AllowOverride Options" or "AllowOverride All" privileges)

With PHP 4.0, there are several Apache directives that allow you to change the PHP configuration from within the Apache configuration files. For a listing of which directives are **PHP_INI_ALL**, **PHP_INI_PERDIR**, or **PHP_INI_SYSTEM**, have a look at the table found within the **ini_set()** documentation.

Note: With PHP 3.0, there are Apache directives that correspond to each configuration setting in the php3.ini name, except the name is prefixed by "php3_".

php_value name value

Sets the value of the specified directive. Can be used only with PHP_INI_ALL and PHP_INI_PERDIR type directives. To clear a previously set value use none as the value.

```
php_value auto_prepend_file none
```

php_flag name on|off

Used to set a Boolean configuration directive. Can be used only with PHP_INI_ALL and PHP_INI_PERDIR type directives.

php_admin_value name value

Sets the value of the specified directive. This can NOT be used in .htaccess files. Any directive type set with php_admin_value can not be overridden by .htaccess or virtualhost directives.

php_admin_flag name on|off

Used to set a Boolean configuration directive. This can NOT be used in .htaccess files. Any directive type set with php_admin_flag can not be overridden by .htaccess or virtualhost directives.

Used to set a Boolean configuration directive. This can NOT be used in .htaccess files. Any directive type set with php_admin_flag can not be overridden by .htaccess or virtualhost directives.

Example 4-2. Apache configuration example

```
<IfModule mod_php4.c>
  php_value include_path ".:/usr/local/lib/php"
  php_admin_flag safe_mode on
</IfModule>
<IfModule mod_php3.c>
  php3_include_path ".:/usr/local/lib/php"
  php3_safe_mode on
</IfModule>
```

Note: PHP constants do not exist outside of PHP. For example, in httpd.conf you can not use PHP constants such as **E_ALL** or **E_NOTICE** to set the error_reporting directive as they will have no meaning and will evaluate to 0. Use the associated bitmask values instead. These constants can be used in php.ini

---

Other interfaces to PHP

Regardless of the interface to PHP you can change certain values at runtime of your scripts through **ini_set()**. The following table provides an overview at which level a directive can be set/changed.

Table 4-1. Definition of PHP_INI_* constants

Constant	Value	Meaning
PHP_INI_USER	1	Entry can be set in user scripts
PHP_INI_PERDIR	2	Entry can be set in php.ini, .htaccess or httpd.conf
PHP_INI_SYSTEM	4	Entry can be set in php.ini or httpd.conf
PHP_INI_ALL	7	Entry can be set anywhere

You can view the settings of the configuration values in the output of **phpinfo()**. You can also access the values of individual configuration directives using **ini_get()** or **get_cfg_var()**.

Configuration directives

Httpd Options

Table 4-2. Httpd Options

Name	Default	Changeable
async_send	"0"	PHP_INI_ALL

Language Options

Table 4-3. Language and Misc Configuration Options

Name	Default	Changeable
short_open_tag	On	PHP_INI_SYSTEM|PHP_INI_PERDIR
asp_tags	Off	PHP_INI_SYSTEM|PHP_INI_PERDIR
precision	"14"	PHP_INI_ALL
y2k_compliance	Off	PHP_INI_ALL
allow_call_time_pass_reference	On	PHP_INI_SYSTEM|PHP_INI_PERDIR
expose_php	On	PHP_INI_SYSTEM

Here is a short explanation of the configuration directives.

short_open_tag **boolean**

Tells whether the short form (**<? ?>**) of PHP's open tag should be allowed. If you want to use PHP in combination with XML, you can disable this option in order to use **<?xml ?>** inline. Otherwise, you can print it with PHP, for example: **<?php echo '<?xml version="1.0"'; ?>**. Also if disabled, you must use the long form of the PHP open tag (**<?php ?>**).

Note: This directive also affects the shorthand **<?=**, which is identical to **<? echo**. Use of this shortcut requires short_open_tag to be on.

asp_tags **boolean**

Enables the use of ASP-like <% %> tags in addition to the usual <?php ?> tags. This includes the variable-value printing shorthand of <%= $value %>. For more information, see Escaping from HTML.

Note: Support for ASP-style tags was added in 3.0.4.

precision **integer**

The number of significant digits displayed in floating point numbers.

y2k_compliance **boolean**

Enforce year 2000 compliance (will cause problems with non-compliant browsers)

allow_call_time_pass_reference **boolean**

Whether to enable the ability to force arguments to be passed by reference at function call time. This method is deprecated and is likely to be unsupported in future versions of PHP/Zend. The encouraged method of specifying which arguments should be passed by reference is in the function declaration. You're encouraged to try and turn this option Off and make sure your scripts work properly with it in order to ensure they will work with future versions of the language

deprecated and is likely to be unsupported in future versions of PHP/Zend. The encouraged method of specifying which arguments should be passed by reference is in the function declaration. You're encouraged to try and turn this option Off and make sure your scripts work properly with it in order to ensure they will work with future versions of the language (you will receive a warning each time you use this feature, and the argument will be passed by value instead of by reference).

See also References Explained.

expose_php **boolean**

Decides whether PHP may expose the fact that it is installed on the server (e.g. by adding its signature to the Web server header). It is no security threat in any way, but it makes it possible to determine whether you use PHP on your server or not.

Resource Limits

Table 4-4. Resource Limits

Name	Default	Changeable
memory_limit	"8M"	PHP_INI_ALL

Here is a short explanation of the configuration directives.

memory_limit **integer**

This sets the maximum amount of memory in bytes that a script is allowed to allocate. This helps prevent poorly written scripts for eating up all available memory on a server. In order to use this directive you must have enabled it at compile time. So, your configure line would have included: --enable-memory-limit. Note that you have to set it to -1 if you don't want any limit for your memory.

See also: max_execution_time.

Data Handling

Table 4-5. Data Handling Configuration Options

Name	Default	Changeable
track-vars	"On"	PHP_INI_??
arg_separator.output	"&"	PHP_INI_ALL
arg_separator.input	"&"	PHP_INI_SYSTEM|PHP_INI_PERDIR
variables_order	"EGPCS"	PHP_INI_ALL
register_globals	"Off"	PHP_INI_PERDIR|PHP_INI_SYSTEM
register_argc_argv	"On"	PHP_INI_PERDIR|PHP_INI_SYSTEM
post_max_size	"8M"	PHP_INI_SYSTEM|PHP_INI_PERDIR
gpc_order	"GPC"	PHP_INI_ALL
auto_prepend_file	""	PHP_INI_SYSTEM|PHP_INI_PERDIR
auto_append_file	""	PHP_INI_SYSTEM|PHP_INI_PERDIR
default_mimetype	"text/html"	PHP_INI_ALL
default_charset	"iso-8859-1"	PHP_INI_ALL
always_populate_raw_post_data	"0"	PHP_INI_SYSTEM|PHP_INI_PERDIR
allow_webdav_methods	"0"	PHP_INI_SYSTEM|PHP_INI_PERDIR

Here is a short explanation of the configuration directives.

track_vars **boolean**

If enabled, then Environment, GET, POST, Cookie, and Server variables can be found in the global associative arrays $_ENV, $_GET, $_POST, $_COOKIE, and $_SERVER.

Note that as of PHP 4.0.3, track_vars is always turned on.

arg_separator.output **string**

The separator used in PHP generated URLs to separate arguments.

arg_separator.input **string**

List of separator(s) used by PHP to parse input URLs into variables.

Note: Every character in this directive is considered as separator!

variables_order **string**

Set the order of the EGPCS (Environment, GET, POST, Cookie, Server) variable parsing. The default setting of this directive is "EGPCS". Setting this to "GP", for example, will cause PHP to completely ignore environment variables, cookies and server variables, and to overwrite any GET method variables with POST-method variables of the same name.

See also register_globals.

register_globals **boolean**

Tells whether or not to register the EGPCS (Environment, GET, POST, Cookie, Server) variables as global variables. For example: if register_globals = on, the url http://www.example.com/test.php?id=3 will produce $id. Or, $DOCUMENT_ROOT from $_SERVER['DOCUMENT_ROOT']. You may want to turn this off if you don't want to clutter your scripts' global scope with user data. As of PHP 4.2.0, this directive defaults to off. It's preferred to go through PHP Predefined Variables instead, such as the superglobals: $_ENV, $_GET, $_POST, $_COOKIE, and $_SERVER. Please read the security chapter on Using register_globals for related information.

Please note that register_globals cannot be set at runtime (**ini_set()**). Although, you can use .htaccess if your host allows it as described above. An example .htaccess entry: **php_flag register_globals on**.

Note: register_globals is affected by the variables_order directive.

register_argc_argv **boolean**

Note: register_globals is affected by the variables_order directive.

register_argc_argv **boolean**

Tells PHP whether to declare the argv & argc variables (that would contain the GET information).

See also command line. Also, this directive became available in PHP 4.0.0 and was always "on" before that.

post_max_size **integer**

Sets max size of post data allowed. This setting also affects file upload. To upload large files, this value must be larger than upload_max_filesize.

If memory limit is enabled by your configure script, memory_limit also affects file uploading. Generally speaking, memory_limit should be larger than post_max_size.

gpc_order **string**

Set the order of GET/POST/COOKIE variable parsing. The default setting of this directive is "GPC". Setting this to "GP", for example, will cause PHP to completely ignore cookies and to overwrite any GET method variables with POST-method variables of the same name.

Note: This option is not available in PHP 4. Use variables_order instead.

auto_prepend_file **string**

Specifies the name of a file that is automatically parsed before the main file. The file is included as if it was called with the **include()** function, so include_path is used.

The special value none disables auto-prepending.

auto_append_file **string**

Specifies the name of a file that is automatically parsed after the main file. The file is included as if it was called with the **include()** function, so include_path is used.

The special value none disables auto-appending.

Note: If the script is terminated with **exit()**, auto-append will not occur.

default_mimetype **string**

default_charset **string**

As of 4.0b4, PHP always outputs a character encoding by default in the Content-type: header. To disable sending of the charset, simply set it to be empty.

always_populate_raw_post_data **boolean**

Always populate the $HTTP_RAW_POST_DATA variable.

allow_webdav_methods **boolean**

Allow handling of WebDAV http requests within PHP scripts (eg. PROPFIND, PROPPATCH, MOVE, COPY, etc..) If you want to get the post data of those requests, you have to set always_populate_raw_post_data as well.

See also: magic_quotes_gpc, magic-quotes-runtime, and magic_quotes_sybase.

Paths and Directories

Table 4-6. Paths and Directories Configuration Options

Name	Default	Changeable
include_path	PHP_INCLUDE_PATH	PHP_INI_ALL
doc_root	PHP_INCLUDE_PATH	PHP_INI_SYSTEM
user_dir	NULL	PHP_INI_SYSTEM
extension_dir	PHP_EXTENSION_DIR	PHP_INI_SYSTEM
cgi.force_redirect	"1"	PHP_INI_SYSTEM
cgi.redirect_status_env	""	PHP_INI_SYSTEM
fastcgi.impersonate	"0"	PHP_INI_SYSTEM

Here is a short explanation of the configuration directives.

include_path **string**

Specifies a list of directories where the **require()**, **include()** and **fopen_with_path()** functions look for files. The format is like the system's PATH environment variable: a list of directories separated with a colon in UNIX or semicolon in Windows.

Example 4-3. UNIX include_path

```
include_path=".:/php/includes"
```

Example 4-4. Windows include_path

```
include_path=".;c:\php\includes"
```

Using a . in the include path allows for relative includes as it means the current directory.

doc_root **string**

PHP's "root directory" on the server. Only used if non-empty. If PHP is configured with safe mode, no files outside this directory are served. If PHP was not compiled with FORCE_REDIRECT, you SHOULD set doc_root if you are running php as a CGI under any web server (other than IIS) The alternative is to use the cgi.force_redirect configuration below.

user_dir **string**

The base name of the directory used on a user's home directory for PHP files, for example public_html.

extension_dir **string**

In what directory PHP should look for dynamically loadable extensions. See also: enable_dl, and **dl()**.

extension **string**

In what directory PHP should look for dynamically loadable extensions. See also: enable_dl, and dl().

extension **string**

Which dynamically loadable extensions to load when PHP starts up.

cgi.force_redirect **boolean**

cgi.force_redirect is necessary to provide security running PHP as a CGI under most web servers. Left undefined, PHP turns this on by default. You can turn it off AT YOUR OWN RISK.

Note: Windows Users: You CAN safely turn this off for IIS, in fact, you MUST. To get OmniHTTPD or Xitami to work you MUST turn it off.

cgi.cgi.redirect_status_env **string**

If cgi.force_redirect is turned on, and you are not running under Apache or Netscape (iPlanet) web servers, you MAY need to set an environment variable name that PHP will look for to know it is OK to continue execution.

Note: Setting this variable MAY cause security issues, KNOW WHAT YOU ARE DOING FIRST.

fastcgi.impersonate **string**

FastCGI under IIS (on WINNT based OS) supports the ability to impersonate security tokens of the calling client. This allows IIS to define the security context that the request runs under. mod_fastcgi under Apache does not currently support this feature (03/17/2002) Set to 1 if running under IIS. Default is zero.

File Uploads

Table 4-7. File Uploads Configuration Options

Name	Default	Changeable
file_uploads	"1"	PHP_INI_SYSTEM
upload_tmp_dir	NULL	PHP_INI_SYSTEM
upload_max_filesize	"2M"	PHP_INI_SYSTEM|PHP_INI_PERDIR

Here is a short explanation of the configuration directives.

file_uploads **boolean**

Whether or not to allow HTTP file uploads. See also the upload_max_filesize, upload_tmp_dir, and post_max_size directives.

upload_tmp_dir **string**

The temporary directory used for storing files when doing file upload. Must be writable by whatever user PHP is running as. If not specified PHP will use the system's default.

upload_max_filesize **integer**

The maximum size of an uploaded file.

General SQL

Table 4-8. General SQL Configuration Options

Name	Default	Changeable
sql.safe_mode	"0"	PHP_INI_SYSTEM

Here is a short explanation of the configuration directives.

sql.safe_mode **boolean**

Debugger Configuration Directives

debugger.host **string**

DNS name or IP address of host used by the debugger.

debugger.port **string**

Port number used by the debugger.

debugger.enabled **boolean**

Whether the debugger is enabled.

Chapter 5. Security

PHP is a powerful language and the interpreter, whether included in a web server as a module or executed as a separate CGI binary, is able to access files, execute commands and open network connections on the server. These properties make anything run on a web server insecure by default. PHP is designed specifically to be a more secure language for writing CGI programs than Perl or C, and with correct selection of compile-time and runtime configuration options, and proper coding practices, it can give you exactly the combination of freedom and security you need.

As there are many different ways of utilizing PHP, there are many configuration options controlling its behaviour. A large selection of options guarantees you can use PHP for a lot of purposes, but it also means there are combinations of these options and server configurations that result in an insecure setup.

The configuration flexibility of PHP is equally rivalled by the code flexibility. PHP can be used to build complete server applications, with all the power of a shell user, or it can be used for simple server-side includes with little risk in a tightly controlled environment. How you build that environment, and how secure it is, is largely up to the PHP developer.

This chapter starts with some general security advice, explains the different configuration option combinations and the situations they can be safely used, and describes different considerations in coding for different levels of security.

General considerations

General considerations

A completely secure system is a virtual impossibility, so an approach often used in the security profession is one of balancing risk and usability. If every variable submitted by a user required two forms of biometric validation (such as a retinal scan and a fingerprint), you would have an extremely high level of accountability. It would also take half an hour to fill out a fairly complex form, which would tend to encourage users to find ways of bypassing the security.

The best security is often inobtrusive enough to suit the requirements without the user being prevented from accomplishing their work, or over-burdening the code author with excessive complexity. Indeed, some security attacks are merely exploits of this kind of overly built security, which tends to erode over time.

A phrase worth remembering: A system is only as good as the weakest link in a chain. If all transactions are heavily logged based on time, location, transaction type, etc. but the user is only verified based on a single cookie, the validity of tying the users to the transaction log is severely weakened.

When testing, keep in mind that you will not be able to test all possibilities for even the simplest of pages. The input you may expect will be completely unrelated to the input given by a disgruntled employee, a cracker with months of time on their hands, or a housecat walking across the keyboard. This is why it's best to look at the code from a logical perspective, to discern where unexpected data can be introduced, and then follow how it is modified, reduced, or amplified.

The Internet is filled with people trying to make a name for themselves by breaking your code, crashing your site, posting inappropriate content, and otherwise making your day interesting. It doesn't matter if you have a small or large site, you are a target by simply being online, by having a server that can be connected to. Many cracking programs do not discern by size, they simply trawl massive IP blocks looking for victims. Try not to become one.

Installed as CGI binary

Possible attacks

Using PHP as a CGI binary is an option for setups that for some reason do not wish to integrate PHP as a module into server software (like Apache), or will use PHP with different kinds of CGI wrappers to create safe chroot and setuid environments for scripts. This setup usually involves installing executable PHP binary to the web server cgi-bin directory. CERT advisory CA-96.11 recommends against placing any interpreters into cgi-bin. Even if the PHP binary can be used as a standalone interpreter, PHP is designed to prevent the attacks this setup makes possible:

- Accessing system files: http://my.host/cgi-bin/php?/etc/passwd
  The query information in a url after the question mark (?) is passed as command line arguments to the interpreter by the CGI interface. Usually interpreters open and execute the file specified as the first argument on the command line.
  When invoked as a CGI binary, PHP refuses to interpret the command line arguments.
- Accessing any web document on server: http://my.host/cgi-bin/php/secret/doc.html
  The path information part of the url after the PHP binary name, /secret/doc.html is conventionally used to specify the name of the file to be opened and interpreted by the CGI program. Usually some web server configuration directives (Apache: Action) are used to redirect requests to documents like http://my.host/secret/script.php to the PHP interpreter. With this setup, the web server first checks the access permissions to the directory /secret, and after that creates the redirected request http://my.host/cgi-bin/php/secret/script.php. Unfortunately, if the request is originally given in this form, no access checks are made by web server for file /secret/script.php, but only for the /cgi-bin/php file. This way any user able to access /cgi-bin/php is able to access any protected document on the web server.
  In PHP, compile-time configuration option --enable-force-cgi-redirect and runtime configuration directives doc_root and user_dir can be used to prevent this attack, if the server document tree has any directories with access restrictions. See below for full the explanation of the different combinations.

Case 1: only public files served

If your server does not have any content that is not restricted by password or ip based access control, there is no need for these configuration options. If your web server does not allow you to do redirects, or the server does not have a way to communicate to the PHP binary that the request is a safely redirected request, you can specify the option --enable-force-cgi-redirect to the configure script. You still have to make sure your PHP scripts do not rely on one or another way of calling the script, neither by directly http://my.host/cgi-bin/php/dir/script.php nor by redirection http://my.host/dir/script.php.

Redirection can be configured in Apache by using AddHandler and Action directives (see below).

Case 2: using --enable-force-cgi-redirect

This compile-time option prevents anyone from calling PHP directly with a url like http://my.host/cgi-bin/php/secretdir/script.php. Instead, PHP will only parse in this mode if it has gone through a web server redirect rule.

Usually the redirection in the Apache configuration is done with the following directives:

```
Action php-script /cgi-bin/php
AddHandler php-script .php
```

This option has only been tested with the Apache web server, and relies on Apache to set the non-standard CGI environment variable REDIRECT_STATUS on redirected requests. If your web server does not support any way of telling if the request is direct or redirected, you cannot use this option and you must use one of the other ways of running the CGI version documented here.

Case 3: setting doc_root or user_dir

To include active content, like scripts and executables, in the web server document directories is sometimes consider an insecure practice. If, because of some configuration mistake, the scripts are not executed but displayed as regular HTML documents, this may result in leakage of intellectual property or security information like passwords. Therefore many sysadmins will prefer setting up another directory structure for scripts that are accessible only through the PHP CGI, and therefore always interpreted and not displayed as such.

Also if the method for making sure the requests are not redirected, as described in the previous section, is not available, it is necessary to set up a script doc_root that is different from web document root.

You can set the PHP script document root by the configuration directive doc_root in the configuration file, or you can set the environment variable PHP_DOCUMENT_ROOT. If it is set, the CGI version of PHP will always construct the file name to open with this doc_root and the path information in the request, so you can be sure no script is executed outside this directory (except for user_dir below).

with this doc_root and the path information in the request, so you can be sure no script is executed outside this directory (except for user_dir below).

Another option usable here is user_dir. When user_dir is unset, only thing controlling the opened file name is doc_root. Opening an url like http://my.host/~user/doc.php does not result in opening a file under users home directory, but a file called ~user/doc.php under doc_root (yes, a directory name starting with a tilde [~]).

If user_dir is set to for example public_php, a request like http://my.host/~user/doc.php will open a file called doc.php under the directory named public_php under the home directory of the user. If the home of the user is /home/user, the file executed is /home/user/public_php/doc.php.

user_dir expansion happens regardless of the doc_root setting, so you can control the document root and user directory access separately.

Case 4: PHP parser outside of web tree

A very secure option is to put the PHP parser binary somewhere outside of the web tree of files. In /usr/local/bin, for example. The only real downside to this option is that you will now have to put a line similar to:

```
#!/usr/local/bin/php
```

as the first line of any file containing PHP tags. You will also need to make the file executable. That is, treat it exactly as you would treat any other CGI script written in Perl or sh or any other common scripting language which uses the #! shell-escape mechanism for launching itself.

To get PHP to handle PATH_INFO and PATH_TRANSLATED information correctly with this setup, the php parser should be compiled with the --enable-discard-path configure option.

Installed as an Apache module

When PHP is used as an Apache module it inherits Apache's user permissions (typically those of the "nobody" user). This has several impacts on security and authorization. For example, if you are using PHP to access a database, unless that database has built-in access control, you will have to make the database accessable to the "nobody" user. This means a malicious script could access and modify the database, even without a username and password. It's entirely possible that a web spider could stumble across a database administrator's web page, and drop all of your databases. You can protect against this with Apache authorization, or you can design your own access model using LDAP, .htaccess files, etc. and include that code as part of your PHP scripts.

Often, once security is established to the point where the PHP user (in this case, the apache user) has very little risk attached to it, it is discovered that PHP is now prevented from writing any files to user directories. Or perhaps it has been prevented from accessing or changing databases. It has equally been secured from writing good and bad files, or entering good and bad database transactions.

A frequent security mistake made at this point is to allow apache root permissions, or to escalate apache's abilitites in some other way.

Escalating the Apache user's permissions to root is extremely dangerous and may compromise the entire system, so sudo'ing, chroot'ing, or otherwise running as root should not be considered by those who are not security professionals.

There are some simpler solutions. By using open_basedir you can control and restrict what directories are allowed to be used for PHP. You can also set up apache-only areas, to restrict all web based activity to non-user, or non-system, files.

Filesystem Security

PHP is subject to the security built into most server systems with respect to permissions on a file and directory basis. This allows you to control which files in the filesystem may be read. Care should be taken with any files which are world readable to ensure that they are safe for reading by all users who have access to that filesystem.

Since PHP was designed to allow user level access to the filesystem, it's entirely possible to write a PHP script that will allow you to read system files such as /etc/passwd, modify your ethernet connections, send massive printer jobs out, etc. This has some obvious implications, in that you need to ensure that the files that you read from and write to are the appropriate ones.

Consider the following script, where a user indicates that they'd like to delete a file in their home directory. This assumes a situation where a PHP web interface is regularly used for file management, so the Apache user is allowed to delete files in the user home directories.

Example 5-1. Poor variable checking leads to....

```
<?php
// remove a file from the user's home directory
$username = $_POST['user_submitted_name'];
$homedir = "/home/$username";
$file_to_delete = "$userfile";
unlink ("$homedir/$userfile");
echo "$file_to_delete has been deleted!";
?>
```

Since the username is postable from a user form, they can submit a username and file belonging to someone else, and delete files. In this case, you'd want to use some other form of authentication. Consider what could happen if the variables submitted were "../etc/" and "passwd". The code would then effectively read:

Example 5-2. ... A filesystem attack

```
<?php
// removes a file from anywhere on the hard drive that
// the PHP user has access to. If PHP has root access:
$username = "../etc/";
$homedir = "/home/../etc/";
$file_to_delete = "passwd";
unlink ("/home/../etc/passwd");
echo "/home/../etc/passwd has been deleted!";
?>
```

There are two important measures you should take to prevent these issues.

There are two important measures you should take to prevent these issues.

- Only allow limited permissions to the PHP web user binary.
- Check all variables which are submitted.

Here is an improved script:

Example 5-3. More secure file name checking

```
<?php
// removes a file from the hard drive that
// the PHP user has access to.
$username = $_SERVER['REMOTE_USER']; // using an authentication mechanisim

$homedir = "/home/$username";

$file_to_delete = basename("$userfile"); // strip paths
unlink ($homedir/$file_to_delete);

$fp = fopen("/home/logging/filedelete.log","+a"); //log the deletion
$logstring = "$username $homedir $file_to_delete";
fputs ($fp, $logstring);
fclose($fp);

echo "$file_to_delete has been deleted!";
?>
```

However, even this is not without it's flaws. If your authentication system allowed users to create their own user logins, and a user chose the login "../etc/", the system is once again exposed. For this reason, you may prefer to write a more customized check:

Example 5-4. More secure file name checking

```
<?php
$username = $_SERVER['REMOTE_USER']; // using an authentication mechanisim
$homedir = "/home/$username";

if (!ereg('^[^./][^/]*$', $userfile))
    die('bad filename'); //die, do not process

if (!ereg('^[^./][^/]*$', $username))
    die('bad username'); //die, do not process
//etc...
?>
```

Depending on your operating system, there are a wide variety of files which you should be concerned about, including device entries (/dev/ or COM1), configuration files (/etc/ files and the .ini files), well known file storage areas (/home/, My Documents), etc. For this reason, it's usually easier to create a policy where you forbid everything except for what you explicitly allow.

Database Security

Nowadays, databases are cardinal components of any web based application by enabling websites to provide varying dynamic content. Since very sensitive or secret informations can be stored in such database, you should strongly consider to protect them somehow.

To retrieve or to store any information you need to connect to the database, send a legitimate query, fetch the result, and close the connecion. Nowadays, the commonly used query language in this interaction is the Structured Query Language (SQL). See how an attacker can tamper with an SQL query.

As you can realize, PHP cannot protect your database by itself. The following sections aim to be an introduction into the very basics of how to access and manipulate databases within PHP scripts.

Keep in mind this simple rule: defence in depth. In the more place you take the more action to increase the protection of your database, the less probability of that an attacker succeeds, and exposes or abuse any stored secret information. Good design of the database schema and the application deals with your greatest fears.

Designing Databases

The first step is always to create the database, unless you want to use an existing third party's one. When a database is created, it is assigned to an owner, who executed the creation statement. Usually, only the owner (or a superuser) can do anything with the objects in that database, and in order to allow other users to use it, privileges must be granted.

Applications should never connect to the database as its owner or a superuser, because these users can execute any query at will, for example, modifying the schema (e.g. dropping tables) or deleting its entire content.

You may create different database users for every aspect of your application with very limited rights to database objects. The most required privileges should be granted only, and avoid that the same user can interact with the database in different use cases. This means that if intruders gain access to your database using one of these credentials, they can only effect as many changes as your application can.

You are encouraged not to implement all the business logic in the web application (i.e. your script), instead to do it in the database schema using views, triggers or rules. If the system evolves, new ports will be intended to open to the database, and you have to reimplement the logic in each separate database client. Over and above, triggers can be used to transparently and automatically handle fields, which often provides insight when debugging problems with your application or tracing back transactions.

Connecting to Database

You may want to estabilish the connections over SSL to encrypt client/server communications for increased security, or you can use ssh to encrypt the network connection between clients and the database server. If either of them is done, then monitoring your traffic and gaining informations in this way will be a hard work.

Encrypted Storage Model

SSL/SSH protects data travelling from the client to the server, SSL/SSH does not protect the persistent data stored in a database. SSL is an on-the-wire protocol.

SSL/SSH protects data travelling from the client to the server, SSL/SSH does not protect the persistent data stored in a database. SSL is an on-the-wire protocol.

Once an attacker gains access to your database directly (bypassing the webserver), the stored sensitive data may be exposed or misused, unless the information is protected by the database itself. Encrypting the data is a good way to mitigate this threat, but very few databases offer this type of data encryption.

The easiest way to work around this problem is to first create your own encryption package, and then use it from within your PHP scripts. PHP can assist you in this case with its several extensions, such as Mcrypt and Mhash, covering a wide variety of encryption algorithms. The script encrypts the data be stored first, and decrypts it when retrieving. See the references for further examples how encryption works.

In case of truly hidden data, if its raw representation is not needed (i.e. not be displayed), hashing may be also taken into consideration. The well-known example for the hashing is storing the MD5 hash of a password in a database, instead of the password itself. See also **crypt()** and **md5()**.

Example 5-5. Using hashed password field

```
// storing password hash
$query  = sprintf("INSERT INTO users(name,pwd) VALUES('%s','%s');",
          addslashes($username), md5($password));
$result = pg_exec($connection, $query);

// querying if user submitted the right password
$query = sprintf("SELECT 1 FROM users WHERE name='%s' AND pwd='%s';",
          addslashes($username), md5($password));
$result = pg_exec($connection, $query);

if (pg_numrows($result) > 0) {
    echo "Welcome, $username!";
}
else {
    echo "Authentication failed for $username.";
}
```

SQL Injection

Many web developers are unaware of how SQL queries can be tampered with, and assume that an SQL query is a trusted command. It means that SQL queries are able to circumvent access controls, thereby bypassing standard authentication and authorization checks, and sometimes SQL queries even may allow access to host operating system level commands.

Direct SQL Command Injection is a technique where an attacker creates or alters existing SQL commands to expose hidden data, or to override valuable ones, or even to execute dangerous system level commands on the database host. This is accomplished by the application taking user input and combining it with static parameters to build a SQL query. The following examples are based on true stories, unfortunately.

Owing to the lack of input validation and connecting to the database on behalf of a superuser or the one who can create users, the attacker may create a superuser in your database.

Example 5-6. Splitting the result set into pages ... and making superusers (PostgreSQL and MySQL)

```
$offset = argv[0]; // beware, no input validation!
$query  = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";
// with PostgreSQL
$result = pg_exec($conn, $query);
// with MySQL
$result = mysql_query($query);
```

Normal users click on the 'next', 'prev' links where the $offset is encoded into the URL. The script expects that the incoming $offset is decimal number. However, someone tries to break in with appending **urlencode()**'d form of the following to the URL

```
// in case of PostgreSQL
0;
insert into pg_shadow(usename,usesysid,usesuper,usecatupd,passwd)
    select 'crack', usesysid, 't','t','crack'
    from pg_shadow where usename='postgres';
--

// in case of MySQL
0;
UPDATE user SET Password=PASSWORD('crack') WHERE user='root';
FLUSH PRIVILEGES;
```

If it happened, then the script would present a superuser access to him. Note that 0; is to supply a valid offset to the original query and to terminate it.

> **Note:** It is common technique to force the SQL parser to ignore the rest of the query written by the developer with -- which is the comment sign in SQL.

A feasible way to gain passwords is to circumvent your search result pages. What the attacker needs only is to try if there is any submitted variable used in SQL statement which is not handled properly. These filters can be set commonly in a preceding form to customize WHERE, ORDER BY, LIMIT and OFFSET clauses in SELECT statements. If your database supports the UNION construct, the attacker may try to append an entire query to the original one to list passwords from an arbitrary table. Using encrypted password fields is strongly encouraged.

Example 5-7. Listing out articles ... and some passwords (any database server)

```
$query  = "SELECT id, name, inserted, size FROM products
            WHERE size = '$size'
            ORDER BY $order LIMIT $limit, $offset;";
$result = odbc_exec($conn, $query);
```

The static part of the query can be combined with another SELECT statement which reveals all passwords:

```
'
union select '1', concat(uname||'-'||passwd) as name, '1971-01-01', '0' from usertable;
--
```

If this query (playing with the ' and --) were assigned to one of the variables used in $query, the query beast awakened.

SQL UPDATEs are also subject to attacking your database. These queries are also threatened by chopping and appending an entirely new query to it. But the attacker might fiddle with the SET clause. In this case some schema information must be possessed to manipulate the query successfully. This can be acquired by examing the form variable names, or just simply brute forcing. There are not so many naming convention for fields storing passwords or usernames.

Example 5-8. From resetting a password ... to gaining more privileges (any database server)

```
$query = "UPDATE usertable SET pwd='$pwd' WHERE uid='$uid';";
```

But a malicious user sumbits the value ' or uid like'%admin%'; -- to $uid to change the admin's password, or simply sets $pwd to "hehehe', admin='yes', trusted=100 " (with a trailing space) to gain more privileges. Then, the query will be twisted:

```
// $uid == ' or uid like'%admin%'; --
$query = "UPDATE usertable SET pwd='...' WHERE uid='' or uid like '%admin%'; --";

// $pwd == "hehehe', admin='yes', trusted=100 "
$query = "UPDATE usertable SET pwd='hehehe', admin='yes', trusted=100 WHERE ...;"
```

A frightening example how operating system level commands can be accessed on some database hosts.

Example 5-9. Attacking the database host's operating system (MSSQL Server)

```
$query  = "SELECT * FROM products WHERE id LIKE '%$prod%'";
$result = mssql_query($query);
```

If attacker submits the value a%' exec master..xp_cmdshell 'net user test testpass /ADD' -- to $prod, then the $query will be:

```
$query  = "SELECT * FROM products
          WHERE id LIKE '%a%'
          exec master..xp_cmdshell 'net user test testpass /ADD'--";
$result = mssql_query($query);
```

MSSQL Server executes the SQL statements in the batch including a command to add a new user to the local accounts database. If this application were running as sa and the MSSQLSERVER service is running with sufficient privileges, the attacker would now have an account with which to access this machine.

> **Note:** Some of the examples above is tied to a specific database server. This does not mean that a similar attack is impossible against other products. Your database server may be so vulnerable in other manner.

Avoiding techniques

You may plead that the attacker must possess a piece of information about the database schema in most examples. You are right, but you never know when and how it can be taken out, and if it happens, your database may be exposed. If you are using an open source, or publicly available database handling package, which may belong to a content management system or forum, the intruders easily produce a copy of a piece of your code. It may be also a security risk if it is a poorly designed one.

These attacks are mainly based on exploiting the code not being written with security in mind. Never trust on any kind of input, especially which comes from the client side, even though it comes from a select box, a hidden input field or a cookie. The first example shows that such a blameless query can cause disasters.

- Never connect to the database as a superuser or as the database owner. Use always customized users with very limited privileges.
- Check if the given input has the expected data type. PHP has a wide range of input validating functions, from the simplest ones found in Variable Functions and in Character Type Functions (e.g. **is_numeric()**, **ctype_digit()** respectively) onwards the Perl compatible Regular Expressions support.
- If the application waits for numerical input, consider to verify data with **is_numeric()**, or silently change its type using **settype()**, or use its numeric representation by **sprintf()**.

  Example 5-10. A more secure way to compose a query for paging

  ```
  settype($offset, 'integer');
  $query = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";

  // please note %d in the format string, using %s would be meaningless
  $query = sprintf("SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET %d;",
              $offset);
  ```

- Quote each non numeric user input which is passed to the database with **addslashes()** or **addcslashes()**. See the first example. As the examples shows, quotes burnt into the static part of the query is not enough, and can be easily hacked.
- Do not print out any database specific information, especially about the schema, by fair means or foul. See also Error Reporting and Error Handling and Logging Functions.
- You may use stored procedures and previously defined cursors to abstract data access so that users do not directly access tables or views, but this solution has another impacts.

Besides these, you benefit from logging queries either within your script or by the database itself, if it supports. Obviously, the logging is unable to prevent any harmful attempt, but it can be helpful to trace back which application has been circumvented. The log is not useful by itself, but through the information it contains. The more detail is generally better.

Error Reporting

With PHP security, there are two sides to error reporting. One is beneficial to increasing security, the other is detrimental.

A standard attack tactic involves profiling a system by feeding it improper data, and checking for the kinds, and contexts, of the errors which are returned. This allows the system cracker to probe for information about the server, to determine possible weaknesses. For example, if an attacker had gleaned information about a page based on a prior form submission, they may attempt to override variables, or modify them:

errors which are returned. This allows the system cracker to probe for information about the server, to determine possible weaknesses. For example, if an attacker had gleaned information about a page based on a prior form submission, they may attempt to override variables, or modify them:

Example 5-11. Attacking Variables with a custom HTML page

```
<form method="post" action="attacktarget?username=badfoo&password=badfoo">
<input type="hidden" name="username" value="badfoo">
<input type="hidden" name="password" value="badfoo">
</form>
```

The PHP errors which are normally returned can be quite helpful to a developer who is trying to debug a script, indicating such things as the function or file that failed, the PHP file it failed in, and the line number which the failure occured in. This is all information that can be exploited. It is not uncommon for a php developer to use **show_source()**, **highlight_string()**, or **highlight_file()** as a debugging measure, but in a live site, this can expose hidden variables, unchecked syntax, and other dangerous information. Especially dangerous is running code from known sources with built-in debugging handlers, or using common debugging techniques. If the attacker can determine what general technique you are using, they may try to brute-force a page, by sending various common debugging strings:

Example 5-12. Exploiting common debugging variables

```
<form method="post" action="attacktarget?errors=Y&amp;showerrors=1"&debug=1">
<input type="hidden" name="errors" value="Y">
<input type="hidden" name="showerrors" value="1">
<input type="hidden" name="debug" value="1">
</form>
```

Regardless of the method of error handling, the ability to probe a system for errors leads to providing an attacker with more information.

For example, the very style of a generic PHP error indicates a system is running PHP. If the attacker was looking at an .html page, and wanted to probe for the back-end (to look for known weaknesses in the system), by feeding it the wrong data they may be able to determine that a system was built with PHP.

A function error can indicate whether a system may be running a specific database engine, or give clues as to how a web page or programmed or designed. This allows for deeper investigation into open database ports, or to look for specific bugs or weaknesses in a web page. By feeding different pieces of bad data, for example, an attacker can determine the order of authentication in a script, (from the line number errors) as well as probe for exploits that may be exploited in different locations in the script.

A filesystem or general PHP error can indicate what permissions the webserver has, as well as the structure and organization of files on the web server. Developer written error code can aggravate this problem, leading to easy exploitation of formerly "hidden" information.

There are three major solutions to this issue. The first is to scrutinize all functions, and attempt to compensate for the bulk of the errors. The second is to disable error reporting entirely on the running code. The third is to use PHP's custom error handling functions to create your own error handler. Depending on your security policy, you may find all three to be applicable to your situation.

One way of catching this issue ahead of time is to make use of PHP's own **error_reporting()**, to help you secure your code and find variable usage that may be dangerous. By testing your code, prior to deployment, with E_ALL, you can quickly find areas where your variables may be open to poisoning or modification in other ways. Once you are ready for deployment, by using E_NONE, you insulate your code from probing.

Example 5-13. Finding dangerous variables with E_ALL

```
<?php
if ($username) {  // Not initialized or checked before usage
    $good_login = 1;
}
if ($good_login == 1) { // If above test fails, not initialized or checked before usage
    fpassthru ("/highly/sensitive/data/index.html");
}
?>
```

Using Register Globals

One feature of PHP that can be used to enhance security is configuring PHP with register_globals = off. By turning off the ability for any user-submitted variable to be injected into PHP code, you can reduce the amount of variable poisoning a potential attacker may inflict. They will have to take the additional time to forge submissions, and your internal variables are effectively isolated from user submitted data.

While it does slightly increase the amount of effort required to work with PHP, it has been argued that the benefits far outweigh the effort.

Example 5-14. Working with register_globals=on

```
<?php
if ($username) {  // can be forged by a user in get/post/cookies
    $good_login = 1;
}

if ($good_login == 1) { // can be forged by a user in get/post/cookies,
    fpassthru ("/highly/sensitive/data/index.html");
}
?>
```

Example 5-15. Working with register_globals = off

```
<?php
if($_COOKIE['username']){
    // can only come from a cookie, forged or otherwise
    $good_login = 1;
    fpassthru ("/highly/sensitive/data/index.html");
}
```

```
    fpassthru ("/highly/sensitive/data/index.html");
}
?>
```

By using this wisely, it's even possible to take preventative measures to warn when forging is being attempted. If you know ahead of time exactly where a variable should be coming from, you can check to see if submitted data is coming from an inappropriate kind of submission. While it doesn't guarantee that data has not been forged, it does require an attacker to guess the right kind of forging.

Example 5-16. Detecting simple variable poisoning

```
<?php
if ($_COOKIE['username'] &&
    !$_POST['username'] &&
    !$_GET['username'] ) {
    // Perform other checks to validate the user name...
    $good_login = 1;
    fpassthru ("/highly/sensitive/data/index.html");
} else {
    mail("admin@example.com", "Possible breakin attempt", $_SERVER['REMOTE_ADDR']);
    echo "Security violation, admin has been alerted.";
    exit;
}
?>
```

Of course, simply turning off register_globals does not mean code is secure. For every piece of data that is submitted, it should also be checked in other ways.

User Submitted Data

The greatest weakness in many PHP programs is not inherent in the language itself, but merely an issue of code not being written with security in mind. For this reason, you should always take the time to consider the implications of a given piece of code, to ascertain the possible damage if an unexpected variable is submitted to it.

Example 5-17. Dangerous Variable Usage

```
<?php
// remove a file from the user's home directory... or maybe
// somebody else's?
unlink ($evil_var);

// Write logging of their access... or maybe an /etc/passwd entry?
fputs ($fp, $evil_var);

// Execute something trivial.. or rm -rf *?
system ($evil_var);
exec ($evil_var);

?>
```

You should always carefully examine your code to make sure that any variables being submitted from a web browser are being properly checked, and ask yourself the following questions:

- Will this script only affect the intended files?
- Can unusual or undesirable data be acted upon?
- Can this script be used in unintended ways?
- Can this be used in conjunction with other scripts in a negative manner?
- Will any transactions be adequately logged?

By adequately asking these questions while writing the script, rather than later, you prevent an unfortunate re-write when you need to increase your security. By starting out with this mindset, you won't guarantee the security of your system, but you can help improve it.

You may also want to consider turning off register_globals, magic_quotes, or other convenience settings which may confuse you as to the validity, source, or value of a given variable. Working with PHP in error_reporting(E_ALL) mode can also help warn you about variables being used before they are checked or initialized (so you can prevent unusual data from being operated upon).

Hiding PHP

In general, security by obscurity is one of the weakest forms of security. But in some cases, every little bit of extra security is desirable.

A few simple techniques can help to hide PHP, possibly slowing down an attacker who is attempting to discover weaknesses in your system. By setting expose_php = off in your php.ini file, you reduce the amount of information available to them.

Another tactic is to configure web servers such as apache to parse different filetypes through PHP, either with an .htaccess directive, or in the apache configuration file itself. You can then use misleading file extensions:

Example 5-18. Hiding PHP as another language

```
# Make PHP code look like other code types
AddType application/x-httpd-php .asp .py .pl
```

Or obscure it completely:

Example 5-19. Using unknown types for PHP extensions

```
# Make PHP code look like unknown types
AddType application/x-httpd-php .bop .foo .133t
```

Or hide it as html code, which has a slight performance hit because all html will be parsed through the PHP engine:

Example 5-20. Using html types for PHP extensions

```
# Make all PHP code look like html
AddType application/x-httpd-php .htm .html
```

For this to work effectively, you must rename your PHP files with the above extensions. While it is a form of security through

```
AddType application/x-httpd-php .htm .html
```

For this to work effectively, you must rename your PHP files with the above extensions. While it is a form of security through obscurity, it's a minor preventative measure with few drawbacks.

Keeping Current

PHP, like any other large system, is under constant scrutiny and improvement. Each new version will often include both major and minor changes to enhance and repair security flaws, configuration mishaps, and other issues that will affect the overall security and stability of your system.

Like other system-level scripting languages and programs, the best approach is to update often, and maintain awareness of the latest versions and their changes.

II. Language Reference

Table of Contents

Chapter 6. Basic syntax

Escaping from HTML

When PHP parses a file, it simply passes the text of the file through until it encounters one of the special tags which tell it to start interpreting the text as PHP code. The parser then executes all the code it finds, up until it runs into a PHP closing tag, which tells the parser to just start passing the text through again. This is the mechanism which allows you to embed PHP code inside HTML: everything outside the PHP tags is left utterly alone, while everything inside is parsed as code.

There are four sets of tags which can be used to denote blocks of PHP code. Of these, only two (<?php. . .?> and <script language="php">. . .</script>) are always available; the others can be turned on or off from the php.ini configuration file. While the short-form tags and ASP-style tags may be convenient, they are not as portable as the longer versions. Also, if you intend to embed PHP code in XML or XHTML, you will need to use the <?php. . .?> form to conform to the XML.

The tags supported by PHP are:

Example 6-1. Ways of escaping from HTML

1. <?php echo("if you want to serve XHTML or XML documents, do like this\n"); ?>

2. <? echo ("this is the simplest, an SGML processing instruction\n"); ?>
   <?= expression ?> This is a shortcut for "<? echo expression ?>"

3. <script language="php">
       echo ("some editors (like FrontPage) don't
             like processing instructions");
   </script>

4. <% echo ("You may optionally use ASP-style tags"); %>
   <%= $variable; # This is a shortcut for "<% echo . . ." %>

The first way, <?php. . .?>, is the preferred method, as it allows the use of PHP in XML-conformant code such as XHTML.

The second way is not available always. Short tags are available only when they have been enabled. This can be done via the **short_tags()** function (PHP 3 only), by enabling the short_open_tag configuration setting in the PHP config file, or by compiling PHP with the --enable-short-tags option to **configure**. Even if it is enabled by default in php.ini-dist, use of short tags are discouraged.

The fourth way is only available if ASP-style tags have been enabled using the asp_tags configuration setting.

> **Note:** Support for ASP-style tags was added in 3.0.4.
>
> **Note:** Using short tags should be avoided when developing applications or libraries that are meant for redistribution, or deployment on PHP servers which are not under your control, because short tags may not be supported on the target server. For portable, redistributable code, be sure not to use short tags.

The closing tag for the block will include the immediately trailing newline if one is present. Also, the closing tag automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block.

PHP allows you to use structures like this:

Example 6-2. Advanced escaping

```
<?php
if ($expression) {
   ?>
   <strong>This is true.</strong>
   <?php
} else {
   ?>
   <strong>This is false.</strong>
   <?php
}
?>
```

This works as expected, because when PHP hits the ?> closing tags, it simply starts outputting whatever it finds until it hits

```
?>
```

This works as expected, because when PHP hits the ?> closing tags, it simply starts outputting whatever it finds until it hits another opening tag. The example given here is contrived, of course, but for outputting large blocks of text, dropping out of PHP parsing mode is generally more efficient than sending all of the text through **echo()** or **print()** or somesuch.

Instruction separation

Instructions are separated the same as in C or Perl - terminate each statement with a semicolon.

The closing tag (?>) also implies the end of the statement, so the following are equivalent:

```
<?php
    echo "This is a test";
?>

<?php echo "This is a test" ?>
```

Comments

PHP supports 'C', 'C++' and Unix shell-style comments. For example:

```
<?php
    echo "This is a test"; // This is a one-line c++ style comment
    /* This is a multi line comment
       yet another line of comment */
    echo "This is yet another test";
    echo "One Final Test"; # This is shell-style style comment
?>
```

The "one-line" comment styles actually only comment to the end of the line or the current block of PHP code, whichever comes first.

```
<h1>This is an <?php # echo "simple";?> example.</h1>
<p>The header above will say 'This is an example'.
```

You should be careful not to nest 'C' style comments, which can happen when commenting out large blocks.

```
<?php
 /*
    echo "This is a test"; /* This comment will cause a problem */
 */
?>
```

The one-line comment styles actually only comment to the end of the line or the current block of PHP code, whichever comes first. This means that HTML code after // ?> WILL be printed: ?> skips out of the PHP mode and returns to HTML mode, and // cannot influence that.

Chapter 7. Types

Introduction

PHP supports eight primitive types.

Four scalar types:

- boolean
- integer
- floating-point number (float)
- string

Two compound types:

- array
- object

And finally two special types:

- resource
- NULL

This manual also introduces some pseudo-types for readability reasons:

- mixed
- number
- callback

You may also find some references to the type "double". Consider double the same as float, the two names exist only for historic reasons.

The type of a variable is usually not set by the programmer; rather, it is decided at runtime by PHP depending on the context in which that variable is used.

which that variable is used.

> **Note:** If you want to check out the type and value of a certain expression, use **var_dump()**.

> **Note:** If you simply want a human-readable representation of the type for debugging, use **gettype()**. To check for a certain type, do not use **gettype()**, but use the is_type functions. Some examples:

```
<?php
$bool = TRUE;   // a boolean
$str  = "foo";  // a string
$int  = 12;     // an integer

echo gettype($bool); // prints out "boolean"
echo gettype($str);  // prints out "string"

// If this is an integer, increment it by four
if (is_int($int)) {
    $int += 4;
}

// If $bool is a string, print it out
// (does not print out anything)
if (is_string($bool)) {
    echo "String: $bool";
}
?>
```

If you would like to force a variable to be converted to a certain type, you may either cast the variable or use the **settype()** function on it.

Note that a variable may be evaluated with different values in certain situations, depending on what type it is at the time. For more information, see the section on Type Juggling.

Booleans

This is the easiest type. A **boolean** expresses a truth value. It can be either **TRUE** or **FALSE**.

> **Note:** The boolean type was introduced in PHP 4.

Syntax

To specify a boolean literal, use either the keyword **TRUE** or **FALSE**. Both are case-insensitive.

```
<?php
$foo = True; // assign the value TRUE to $foo
?>
```

Usually you use some kind of operator which returns a **boolean** value, and then pass it on to a control structure.

```
<?php
// == is an operator which test
// equality and returns a boolean
if ($action == "show_version") {
    echo "The version is 1.23";
}

// this is not necessary...
if ($show_separators == TRUE) {
    echo "<hr>\n";
}

// ...because you can simply type
if ($show_separators) {
    echo "<hr>\n";
}
?>
```

Converting to boolean

To explicitly convert a value to **boolean**, use either the (bool) or the (boolean) cast. However, in most cases you do not need to use the cast, since a value will be automatically converted if an operator, function or control structure requires a **boolean** argument.

See also Type Juggling.

When converting to **boolean**, the following values are considered **FALSE**:

- the boolean **FALSE** itself
- the integer 0 (zero)
- the float 0.0 (zero)
- the empty string, and the string "0"
- an array with zero elements
- an object with zero member variables
- the special type NULL (including unset variables)

Every other value is considered **TRUE** (including any resource).

Every other value is considered **TRUE** (including any resource).

Warning
-1 is considered **TRUE**, like any other non-zero (whether negative or positive) number!

```
<?php
echo gettype((bool) "");       // bool(false)
echo gettype((bool) 1);        // bool(true)
echo gettype((bool) -2);       // bool(true)
echo gettype((bool) "foo");    // bool(true)
echo gettype((bool) 2.3e5);    // bool(true)
echo gettype((bool) array(12)); // bool(true)
echo gettype((bool) array());  // bool(false)
?>
```

Integers

An **integer** is a number of the set Z = {..., -2, -1, 0, 1, 2, ...}.

See also: Arbitrary length integers and Floating point numbers

---

Syntax

Integers can be specified in decimal (10-based), hexadecimal (16-based) or octal (8-based) notation, optionally preceded by a sign (- or +).

If you use the octal notation, you must precede the number with a 0 (zero), to use hexadecimal notation precede the number with 0x.

Example 7-1. Integer literals

```
<?php
$a = 1234; # decimal number
$a = -123; # a negative number
$a = 0123; # octal number (equivalent to 83 decimal)
$a = 0x1A; # hexadecimal number (equivalent to 26 decimal)
?>
```

Formally the possible structure for integer literals is:

```
<?php
decimal     : [1-9][0-9]*
            | 0

hexadecimal : 0[xX][0-9a-fA-F]+

octal       : 0[0-7]+

integer     : [+-]?decimal
            | [+-]?hexadecimal
            | [+-]?octal
?>
```

The size of an integer is platform-dependent, although a maximum value of about two billion is the usual value (that's 32 bits signed). PHP does not support unsigned integers.

Integer overflow

If you specify a number beyond the bounds of the **integer** type, it will be interpreted as a **float** instead. Also, if you perform an operation that results in a number beyond the bounds of the **integer** type, a **float** will be returned instead.

```
<?php
$large_number =  2147483647;
var_dump($large_number);
// output: int(2147483647)

$large_number =  2147483648;
var_dump($large_number);
// output: float(2147483648)

// this goes also for hexadecimal specified integers:
var_dump( 0x80000000 );
// output: float(2147483648)

$million = 1000000;
$large_number =  50000 * $million;
var_dump($large_number);
// output: float(50000000000)
?>
```

Warning
Unfortunately, there was a bug in PHP so that this does not always work correctly when there are negative numbers involved. For example: when you do -50000 * $million, the result will be -429496728. However, when both operands are positive there is no problem. This is solved in PHP 4.1.0.

no problem.

This is solved in PHP 4.1.0.

There is no integer division operator in PHP. 1/2 yields the **float** 0.5. You can cast the value to an integer to always round it downwards, or you can use the **round()** function.

```
<?php
var_dump(25/7);         // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7));  // float(4)
?>
```

Converting to integer

To explicitly convert a value to **integer**, use either the (int) or the (integer) cast. However, in most cases you do not need to use the cast, since a value will be automatically converted if an operator, function or control structure requires an **integer** argument. You can also convert a value to integer with the function **intval()**.

See also type-juggling.

From booleans

FALSE will yield 0 (zero), and **TRUE** will yield 1 (one).

From floating point numbers

When converting from float to integer, the number will be rounded towards zero.

If the float is beyond the boundaries of integer (usually +/- 2.15e+9 = 2^31), the result is undefined, since the float hasn't got enough precision to give an exact integer result. No warning, not even a notice will be issued in this case!

Warning

Never cast an unknown fraction to **integer**, as this can sometimes lead to unexpected results.
 	
`<?php`
`echo (int) ( (0.1+0.7) * 10 ); // echoes 7!`
`?>`
 	
See for more information the warning about float-precision.

From strings

See String conversion to numbers

From other types

Caution
Behaviour of converting to integer is undefined for other types. Currently, the behaviour is the same as if the value was first converted to boolean. However, do not rely on this behaviour, as it can change without notice.

Floating point numbers

Floating point numbers (AKA "floats", "doubles" or "real numbers") can be specified using any of the following syntaxes:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Formally:

```
LNUM         [0-9]+
DNUM         ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

The size of a float is platform-dependent, although a maximum of ~1.8e308 with a precision of roughly 14 decimal digits is a common value (that's 64 bit IEEE format).

Floating point precision
It is quite usual that simple decimal fractions like 0.1 or 0.7 cannot be converted into their internal binary counterparts without a little loss of precision. This can lead to confusing results: for example, floor((0.1+0.7)*10) will usually return 7 instead of the expected 8 as the result of the internal representation really being something like 7.9999999999.... This is related to the fact that it is impossible to exactly express some fractions in decimal notation with a finite number of digits. For instance, 1/3 in decimal form becomes 0.3333333. . .. So never trust floating number results to the last digit and never compare floating point numbers for equality. If you really need higher precision, you should use the arbitrary precision math functions or gmp functions instead.

---

need higher precision, you should use the arbitrary precision math functions or gmp functions instead.

Converting to float

For information on when and how strings are converted to floats, see the section titled String conversion to numbers. For values of other types, the conversion is the same as if the value would have been converted to integer and then to float. See the Converting to integer section for more information.

Strings

A **string** is series of characters. In PHP, a character is the same as a byte, that is, there are exactly 256 different characters possible. This also implies that PHP has no native support of Unicode. See **utf8_encode()** and **utf8_decode()** for some Unicode support.

> **Note:** It is no problem for a string to become very large. There is no practical bound to the size of strings imposed by PHP, so there is no reason at all to worry about long strings.

Syntax

A string literal can be specified in three different ways.

- single quoted
- double quoted
- heredoc syntax

Single quoted

The easiest way to specify a simple string is to enclose it in single quotes (the character ').

To specify a literal single quote, you will need to escape it with a backslash (\), like in many other languages. If a backslash needs to occur before a single quote or at the end of the string, you need to double it. Note that if you try to escape any other character, the backslash will also be printed! So usually there is no need to escape the backslash itself.

> **Note:** In PHP 3, a warning will be issued at the E_NOTICE level when this happens.

> **Note:** Unlike the two other syntaxes, variables will not be expanded when they occur in single quoted strings.

```
<?php
echo 'this is a simple string';

echo 'You can also have embedded newlines in
strings this way as it is
okay to do';

// Outputs: "I'll be back"
echo 'Arnold once said: "I\'ll be back"';

// Outputs: You deleted C:\*.*?
echo 'You deleted C:\\*.*?';

// Outputs: You deleted C:\*.*?
echo 'You deleted C:\*.*?';

// Outputs: This will not expand: \n a newline
echo 'This will not expand: \n a newline';

// Outputs: Variables do not $expand $either
echo 'Variables do not $expand $either';
?>
```

Double quoted

If the string is enclosed in double-quotes ("), PHP understands more escape sequences for special characters:

Table 7-1. Escaped characters

sequence	meaning
\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\\	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation

Again, if you try to escape any other character, the backslash will be printed too!

But the most important feature of double-quoted strings is the fact that variable names will be expanded. See string parsing for details.

Heredoc

Another way to delimit strings is by using heredoc syntax ("<<<"). One should provide an identifier after <<<, then the string, and then the same identifier to close the quotation.

The closing identifier must begin in the first column of the line. Also, the identifier used must follow the same naming rules as

then the same identifier to close the quotation.

The closing identifier must begin in the first column of the line. Also, the identifier used must follow the same naming rules as any other label in PHP: it must contain only alphanumeric characters and underscores, and must start with a non-digit character or underscore.

Warning
It is very important to note that the line with the closing identifier contains no other characters, except possibly a semicolon (;). That means especially that the identifier may not be indented, and there may not be any spaces or tabs after or before the semicolon. It's also important to realize that the first character before the closing identifier must be a newline as defined by your operating system. This is \r on Macintosh for example. If this rule is broken and the closing identifier is not "clean" then it's not considered to be a closing identifier and PHP will continue looking for one. If in this case a proper closing identifier is not found then a parse error will result with the line number being at the end of the script.

Heredoc text behaves just like a double-quoted string, without the double-quotes. This means that you do not need to escape quotes in your here docs, but you can still use the escape codes listed above. Variables are expanded, but the same care must be taken when expressing complex variables inside a here doc as with strings.

Example 7-2. Heredoc string quoting example

```
<?php
$str = <<<EOD
Example of string
spanning multiple lines
using heredoc syntax.
EOD;

/* More complex example, with variables. */
class foo
{
    var $foo;
    var $bar;

    function foo()
    {
        $this->foo = 'Foo';
        $this->bar = array('Bar1', 'Bar2', 'Bar3');
    }
}

$foo = new foo();
$name = 'MyName';

echo <<<EOT
My name is "$name". I am printing some $foo->foo.
Now, I am printing some {$foo->bar[1]}.
This should print a capital 'A': \x41
EOT;
?>
```

Note: Heredoc support was added in PHP 4.

Variable parsing

When a string is specified in double quotes or with heredoc, variables are parsed within it.

There are two types of syntax, a simple one and a complex one. The simple syntax is the most common and convenient, it provides a way to parse a variable, an **array** value, or an object property.

The complex syntax was introduced in PHP 4, and can be recognised by the curly braces surrounding the expression.

Simple syntax

If a dollar sign ($) is encountered, the parser will greedily take as much tokens as possible to form a valid variable name. Enclose the variable name in curly braces if you want to explicitly specify the end of the name.

```
<?php
$beer = 'Heineken';
echo "$beer's taste is great"; // works, "'" is an invalid character for varnames
echo "He drank some $beers";   // won't work, 's' is a valid character for varnames
echo "He drank some ${beer}s"; // works
echo "He drank some {$beer}s"; // works
?>
```

Similarly, you can also have an **array** index or an object property parsed. With array indices, the closing square bracket (]) marks the end of the index. For object properties the same rules apply as to simple variables, though with object properties there doesn't exist a trick like the one with variables.

```
<?php
// These examples are specific to using arrays inside of strings.
// When outside of a string, always quote your array string keys
// and do not use {braces} when outside of strings either.

// Let's show all errors
error_reporting(E_ALL);

$fruits = array('strawberry' => 'red', 'banana' => 'yellow');
```

```
error_reporting(E_ALL);

$fruits = array('strawberry' => 'red', 'banana' => 'yellow');

// Works but note that this works differently outside string-quotes
echo "A banana is $fruits[banana].";

// Works
echo "A banana is {$fruits['banana']}.";

// Works but PHP looks for a constant named banana first
// as described below.
echo "A banana is {$fruits[banana]}.";

// Won't work, use braces.  This results in a parse error.
echo "A banana is $fruits['banana'].";

// Works
echo "A banana is " . $fruits['banana'] . ".";

// Works
echo "This square is $square->width meters broad.";

// Won't work. For a solution, see the complex syntax.
echo "This square is $square->width00 centimeters broad.";
?>
```

For anything more complex, you should use the complex syntax.

Complex (curly) syntax

This isn't called complex because the syntax is complex, but because you can include complex expressions this way.

In fact, you can include any value that is in the namespace in strings with this syntax. You simply write the expression the same way as you would outside the string, and then include it in { and }. Since you can't escape '{', this syntax will only be recognised when the $ is immediately following the {. (Use "{\$" or "\{$" to get a literal "{$"). Some examples to make it clear:

```
<?php
// Let's show all errors
error_reporting(E_ALL);

$great = 'fantastic';

// Won't work, outputs: This is { fantastic}
echo "This is { $great}";

// Works, outputs: This is fantastic
echo "This is {$great}";
echo "This is ${great}";

// Works
echo "This square is {$square->width}00 centimeters broad.";

// Works
echo "This works: {$arr[4][3]}";

// This is wrong for the same reason as $foo[bar] is wrong
// outside a string.  In otherwords, it will still work but
// because PHP first looks for a constant named foo, it will
// throw an error of level E_NOTICE (undefined constant).
echo "This is wrong: {$arr[foo][3]}";

// Works.  When using multi-dimensional arrays, always use
// braces around arrays when inside of strings
echo "This works: {$arr['foo'][3]}";

// Works.
echo "This works: " . $arr['foo'][3];

echo "You can even write {$obj->values[3]->name}";

echo "This is the value of the var named $name: {${$name}}";
?>
```

String access by character

Characters within strings may be accessed by specifying the zero-based offset of the desired character after the string in curly braces.

> **Note:** For backwards compatibility, you can still use array-braces for the same purpose. However, this syntax is deprecated as of PHP 4.

Example 7-3. Some string examples

```
<?php
// Get the first character of a string
$str = 'This is a test.';
$first = $str{0};

// Get the third character of a string
```

```
$first = $str{0};

// Get the third character of a string
$third = $str{2};

// Get the last character of a string.
$str = 'This is still a test.';
$last = $str{strlen($str)-1};
?>
```

Useful functions and operators

Strings may be concatenated using the '.' (dot) operator. Note that the '+' (addition) operator will not work for this. Please see String operators for more information.

There are a lot of useful functions for string modification.

See the string functions section for general functions, the regular expression functions for advanced find&replacing (in two tastes: Perl and POSIX extended).

There are also functions for URL-strings, and functions to encrypt/decrypt strings (mcrypt and mhash).

Finally, if you still didn't find what you're looking for, see also the character type functions.

Converting to string

You can convert a value to a string using the (string) cast, or the **strval()** function. String conversion is automatically done in the scope of an expression for you where a string is needed. This happens when you use the **echo()** or **print()** functions, or when you compare a variable value to a string. Reading the manual sections on Types and Type Juggling will make the following clearer. See also **settype()**.

A **boolean** **TRUE** value is converted to the string "1", the **FALSE** value is represented as "" (empty string). This way you can convert back and forth between boolean and string values.

An **integer** or a floating point number (**float**) is converted to a string representing the number with its digits (including the exponent part for floating point numbers).

Arrays are always converted to the string "Array", so you cannot dump out the contents of an **array** with **echo()** or **print()** to see what is inside them. To view one element, you'd do something like echo $arr['foo']. See below for tips on dumping/viewing the entire contents.

Objects are always converted to the string "Object". If you would like to print out the member variable values of an **object** for debugging reasons, read the paragraphs below. If you would like to find out the class name of which an object is an instance of, use **get_class()**.

Resources are always converted to strings with the structure "Resource id #1" where 1 is the unique number of the **resource** assigned by PHP during runtime. If you would like to get the type of the resource, use **get_resource_type()**.

NULL is always converted to an empty string.

As you can see above, printing out the arrays, objects or resources does not provide you any useful information about the values themselfs. Look at the functions **print_r()** and **var_dump()** for better ways to print out values for debugging.

You can also convert PHP values to strings to store them permanently. This method is called serialization, and can be done with the function **serialize()**. You can also serialize PHP values to XML structures, if you have WDDX support in your PHP setup.

String conversion to numbers

When a string is evaluated as a numeric value, the resulting value and type are determined as follows.

The string will evaluate as a **float** if it contains any of the characters '.', 'e', or 'E'. Otherwise, it will evaluate as an integer.

The value is given by the initial portion of the string. If the string starts with valid numeric data, this will be the value used. Otherwise, the value will be 0 (zero). Valid numeric data is an optional sign, followed by one or more digits (optionally containing a decimal point), followed by an optional exponent. The exponent is an 'e' or 'E' followed by one or more digits.

```
<?php
$foo = 1 + "10.5";                // $foo is float (11.5)
$foo = 1 + "-1.3e3";              // $foo is float (-1299)
$foo = 1 + "bob-1.3e3";           // $foo is integer (1)
$foo = 1 + "bob3";                // $foo is integer (1)
$foo = 1 + "10 Small Pigs";       // $foo is integer (11)
$foo = 4 + "10.2 Little Piggies"; // $foo is float (14.2)
$foo = "10.0 pigs " + 1;          // $foo is float (11)
$foo = "10.0 pigs " + 1.0;        // $foo is float (11)
?>
```

For more information on this conversion, see the Unix manual page for strtod(3).

If you would like to test any of the examples in this section, you can cut and paste the examples and insert the following line to see for yourself what's going on:

```
<?php
echo "\$foo==$foo; type is " . gettype ($foo) . "<br />\n";
?>
```

Do not expect to get the code of one character by converting it to integer (as you would do in C for example). Use the functions **ord()** and **chr()** to convert between charcodes and characters.

Arrays

An array in PHP is actually an ordered map. A map is a type that maps values to keys. This type is optimized in several ways, so you can use it as a real array, or a list (vector), hashtable (which is an implementation of a map), dictionary, collection, stack,

An array in PHP is actually an ordered map. A map is a type that maps values to keys. This type is optimized in several ways, so you can use it as a real array, or a list (vector), hashtable (which is an implementation of a map), dictionary, collection, stack, queue and probably more. Because you can have another PHP-array as a value, you can also quite easily simulate trees.

Explanation of those structures is beyond the scope of this manual, but you'll find at least one example for each of those structures. For more information we refer you to external literature about this broad topic.

Syntax

Specifying with array()

An **array** can be created by the **array()** language-construct. It takes a certain number of comma-separated key => value pairs.

```
array( [key =>] value
    , ...
    )
// key is either string or nonnegative integer
// value can be anything
```

```
<?php
$arr = array("foo" => "bar", 12 => true);

echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

A key is either an **integer** or a **string**. If a key is the standard representation of an **integer**, it will be interpreted as such (i.e. "8" will be interpreted as 8, while "08" will be interpreted as "08"). There are no different indexed and associative array types in PHP, there is only one array type, which can both contain integer and string indices.

A value can be of any PHP type.

```
<?php
$arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));

echo $arr["somearray"][6];    // 5
echo $arr["somearray"][13];   // 9
echo $arr["somearray"]["a"];  // 42
?>
```

If you omit a key, the maximum of the integer-indices is taken, and the new key will be that maximum + 1. As integers can be negative, this is also true for negative indices. Having e.g. the highest index being -6 will result in -5 being the new key. If no integer-indices exist yet, the key will be 0 (zero). If you specify a key that already has a value assigned to it, that value will be overwritten.

```
<?php
// This array is the same as ...
array(5 => 43, 32, 56, "b" => 12);

// ...this array
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

Using **TRUE** as a key will evalute to **integer** 1 as key. Using **FALSE** as a key will evalute to **integer** 0 as key. Using NULL as a key will evaluate to an empty string. Using an emptry string as key will create (or overwrite) a key with an empty string and its value, it is not the same as using empty brackets.

You cannot use arrays or objects as keys. Doing so will result in a warning: Illegal offset type.

Creating/modifying with square-bracket syntax

You can also modify an existing array, by explicitly setting values in it.

This is done by assigning values to the array while specifying the key in brackets. You can also omit the key, add an empty pair of brackets ("[]") to the variable-name in that case.

```
$arr[key] = value;
$arr[] = value;
// key is either string or nonnegative integer
// value can be anything
```

If $arr doesn't exist yet, it will be created. So this is also an alternative way to specify an array. To change a certain value, just assign a new value to an element specified with its key. If you want to remove a key/value pair, you need to **unset()** it.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // This is the same as $arr[13] = 56;
                // at this point of the script

$arr["x"] = 42; // This adds a new element to
                // the array with key "x"

unset($arr[5]); // This removes the element from the array
```

```
unset($arr[5]); // This removes the element from the array

unset($arr);    // This deletes the whole array
?>
```

Useful functions

There are quite some useful function for working with arrays, see the array functions section.

> **Note:** The **unset()** function allows unsetting keys of an array. Be aware that the array will NOT be reindexed. If you only use "usual integer indices" (starting from zero, increasing by one), you can achive the reindex effect by using **array_values()**.

```
<?php
$a = array(1 => 'one', 2 => 'two', 3 => 'three');
unset($a[2]);
/* will produce an array that would have been defined as
   $a = array(1 => 'one', 3 => 'three');
   and NOT
   $a = array(1 => 'one', 2 =>'three');
*/

$b = array_values($a);
// Now b is array(1 => 'one', 2 =>'three')
?>
```

The foreach control structure exists specifically for arrays. It provides an easy way to traverse an array.

Array do's and don'ts

Why is $foo[bar] wrong?

You should always use quotes around an associative array index. For example, use $foo['bar'] and not $foo[bar]. But why is $foo[bar] wrong? You might have seen the following syntax in old scripts:

```
<?php
$foo[bar] = 'enemy';
echo $foo[bar];
// etc
?>
```

This is wrong, but it works. Then, why is it wrong? The reason is that this code has an undefined constant (bar) rather than a string ('bar' - notice the quotes), and PHP may in future define constants which, unfortunately for your code, have the same name. It works, because the undefined constant gets converted to a string of the same name automatically for backward compatibility reasons.

More examples to demonstrate this fact:

```
<?php
// Let's show all errors
error_reporting(E_ALL);

$arr = array('fruit' => 'apple', 'veggie' => 'carrot');

// Correct
print $arr['fruit'];  // apple
print $arr['veggie']; // carrot

// Incorrect.  This works but also throws a PHP error of
// level E_NOTICE because of an undefined constant named fruit
//
// Notice: Use of undefined constant fruit - assumed 'fruit' in...
print $arr[fruit];    // apple

// Let's define a constant to demonstrate what's going on.  We
// will assign value 'veggie' to a constant named fruit.
define('fruit','veggie');

// Notice the difference now
print $arr['fruit'];  // apple
print $arr[fruit];    // carrot

// The following is okay as it's inside a string.  Constants are not
// looked for within strings so no E_NOTICE error here
print "Hello $arr[fruit]";      // Hello apple

// With one exception, braces surrounding arrays within strings
// allows constants to be looked for
print "Hello {$arr[fruit]}";    // Hello carrot
print "Hello {$arr['fruit']}";  // Hello apple

// This will not work, results in a parse error such as:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or T_NUM_STRING'
// This of course applies to using autoglobals in strings as well
print "Hello $arr['fruit']";
print "Hello $_GET['foo']";
```

```
print "Hello $_GET['foo']";

// Concatenation is another option
print "Hello " . $arr['fruit']; // Hello apple
?>
```

When you turn **error_reporting()** up to show **E_NOTICE** level errors (such as setting it to **E_ALL**) then you will see these errors. By default, error_reporting is turned down to not show them.

As stated in the syntax section, there must be an expression between the square brackets ('[' and ']'). That means that you can write things like this:

```
<?php
echo $arr[somefunc($bar)];
?>
```

This is an example of using a function return value as the array index. PHP also knows about constants, as you may have seen the E_* ones before.

```
<?php
$error_descriptions[E_ERROR]   = "A fatal error has occured";
$error_descriptions[E_WARNING] = "PHP issued a warning";
$error_descriptions[E_NOTICE]  = "This is just an informal notice";
?>
```

Note that E_ERROR is also a valid identifier, just like bar in the first example. But the last example is in fact the same as writing:

```
<?php
$error_descriptions[1] = "A fatal error has occured";
$error_descriptions[2] = "PHP issued a warning";
$error_descriptions[8] = "This is just an informal notice";
?>
```

because E_ERROR equals 1, etc.

As we already explained in the above examples, $foo[bar] still works but is wrong. It works, because bar is due to its syntax expected to be a constant expression. However, in this case no constant with the name bar exists. PHP now assumes that you meant bar literally, as the string "bar", but that you forgot to write the quotes.

So why is it bad then?

At some point in the future, the PHP team might want to add another constant or keyword, or you may introduce another constant into your application, and then you get in trouble. For example, you already cannot use the words empty and default this way, since they are special reserved keywords.

> **Note:** To reiterate, inside a double-quoted **string**, it's valid to not surround array indexes with quotes so "$foo[bar]" is valid. See the above examples for details on why as well as the section on variable parsing in strings.

Converting to array

For any of the types: **integer**, **float**, **string**, **boolean** and **resource**, if you convert a value to an **array**, you get an array with one element (with index 0), which is the scalar value you started with.

If you convert an **object** to an array, you get the properties (member variables) of that object as the array's elements. The keys are the member variable names.

If you convert a **NULL** value to an array, you get an empty array.

Examples

The array type in PHP is very versatile, so here will be some examples to show you the full power of arrays.

```
<?php
// this
$a = array( 'color' => 'red',
          'taste' => 'sweet',
          'shape' => 'round',
          'name'  => 'apple',
                4       // key will be 0
        );

// is completely equivalent with
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name']  = 'apple';
$a[]        = 4;        // key will be 0

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// will result in the array array(0 => 'a' , 1 => 'b' , 2 => 'c'),
// or simply array('a', 'b', 'c')
?>
```

```
?>
```

Example 7-4. Using array()

```
<?php
// Array as (property-)map
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );

// strictly numerical keys
$array = array( 7,
                8,
                0,
                156,
                -10
              );
// this is the same as array(0 => 7, 1 => 8, ...)

$switching = array(        10, // key = 0
                     5    =>  6,
                     3    =>  7,
                     'a'  =>  4,
                             11, // key = 6 (maximum of integer-indices was 5)
                     '8'  =>  2, // key = 8 (integer!)
                     '02' => 77, // key = '02'
                     0    => 12  // the value 10 will be overwritten by 12
                   );

// empty array
$empty = array();
?>
```

Example 7-5. Collection

```
<?php
$colors = array('red', 'blue', 'green', 'yellow');

foreach ($colors as $color) {
    echo "Do you like $color?\n";
}

/* output:
Do you like red?
Do you like blue?
Do you like green?
Do you like yellow?
*/
?>
```

Note that it is currently not possible to change the values of the array directly in such a loop. A workaround is the following:

Example 7-6. Collection

```
<?php
foreach ($colors as $key => $color) {
    // won't work:
    //$color = strtoupper($color);

    // works:
    $colors[$key] = strtoupper($color);
}
print_r($colors);

/* output:
Array
(
    [0] => RED
    [1] => BLUE
    [2] => GREEN
    [3] => YELLOW
)
*/
?>
```

This example creates a one-based array.

Example 7-7. One-based index

```
<?php
$firstquarter  = array(1 => 'January', 'February', 'March');
print_r($firstquarter);

/* output:
Array
(
    [1] => 'January'
    [2] => 'February'
    [3] => 'March'
)
*/
```

```
)
*/
```

Example 7-8. Filling an array

```
// fill an array with all items from a directory
$handle = opendir('.');
while ($file = readdir($handle)) {
    $files[] = $file;
}
closedir($handle);
?>
```

Arrays are ordered. You can also change the order using various sorting-functions. See the array functions section for more information. You can count the number of items in an array using the **count()** function.

Example 7-9. Sorting array

```
<?php
sort($files);
print_r($files);
?>
```

Because the value of an array can be everything, it can also be another array. This way you can make recursive and multi-dimensional arrays.

Example 7-10. Recursive and multi-dimensional arrays

```
<?php
$fruits = array ( "fruits"  => array ( "a" => "orange",
                                       "b" => "banana",
                                       "c" => "apple"
                                     ),
                  "numbers" => array ( 1,
                                       2,
                                       3,
                                       4,
                                       5,
                                       6,
                                     ),
                  "holes"   => array (      "first",
                                       5 => "second",
                                            "third"
                                     )
                );

// Some examples to address values in the array above
echo $fruits["holes"][5];    // prints "second"
echo $fruits["fruits"]["a"]; // prints "orange"
unset($fruits["holes"][0]);  // remove "first"

// Create a new multi-dimensional array
$juices["apple"]["green"] = "good";
?>
```

You should be aware, that array assignment always involves value copying. You need to use the reference operator to copy an array by reference.

```
<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 is changed,
             // $arr1 is still array(2,3)

$arr3 = &$arr1;
$arr3[] = 4; // now $arr1 and $arr3 are the same
?>
```

Objects

Object Initialization

To initialize an object, you use the new statement to instantiate the object to a variable.

```
<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}

$bar = new foo;
$bar->do_foo();
?>
```

For a full discussion, please read the section Classes and Objects.

Converting to object

If an object is converted to an object, it is not modified. If a value of any other type is converted to an object, a new instace of the stdClass built in class is created. If the value was null, the new instance will be empty. For any other value, a member variable named scalar will contain the value.

```
<?php
$obj = (object) 'ciao';
echo $obj-&gt;scalar;  // outputs 'ciao'
?>
```

Resource

A resource is a special variable, holding a reference to an external resource. Resources are created and used by special functions. See the appendix for a listing of all these functions and the corresponding resource types.

> **Note:** The resource type was introduced in PHP 4

Converting to resource

As resource types hold special handlers to opened files, database connections, image canvas areas and the like, you cannot convert any value to a resource.

Freeing resources

Due to the reference-counting system introduced with PHP4's Zend-engine, it is automatically detected when a resource is no longer referred to (just like Java). When this is the case, all resources that were in use for this resource are made free by the garbage collector. For this reason, it is rarely ever necessary to free the memory manually by using some free_result function.

> **Note:** Persistent database links are special, they are not destroyed by the garbage collector. See also the section about persistent connections.

NULL

The special **NULL** value represents that a variable has no value. **NULL** is the only possible value of type NULL.

> **Note:** The null type was introduced in PHP 4

A variable is considered to be **NULL** if

- it has been assigned the constant **NULL**.
- it has not been set to any value yet.
- it has been **unset()**.

Syntax

There is only one value of type **NULL**, and that is the case-insensitive keyword **NULL**.

```
<?php
$var = NULL;

?>
```

See also **is_null()** and **unset()**.

Pseudo-types used in this documentation

mixed

mixed indicates that a parameter may accept multiple (but not necesseraly all) types.

gettype() for example will accept all PHP types, while **str_replace()** will accept strings and arrays.

---

number

number indicates that a parameter can be either **integer** or **float**.

---

callback

Some functions like **call_user_function()** or **usort()** accept user defined callback functions as a parameter. Callback functions can not only be simple functions but also object methods including static class methods.

A PHP function is simply passed by its name as a string. You can pass any builtin or user defined function with the exception of **array()**, **echo()**, **empty()**, **eval()**, **exit()**, **isset()**, **list()**, **print()** and **unset()**.

A method of an instantiated object is passed as an array containing an object as the element with index 0 and a method name as the element with index 1.

Static class methods can also be passed without instantiating an object of that class by passing the class name instead of an

the element with index 1.

Static class methods can also be passed without instantiating an object of that class by passing the class name instead of an object as the element with index 0.

Example 7-11. Callback function examples

```
<?php

// simple callback example
function foobar() {
    echo "hello world!";
}
call_user_function("foobar");

// method callback examples
class foo {
  function bar() {
    echo "hello world!";
  }
}

$foo = new foo;

call_user_function(array($foo, "bar")); // object method call

call_user_function(array("foo", "bar")); // static class method call

?>
```

Type Juggling

PHP does not require (or support) explicit type definition in variable declaration; a variable's type is determined by the context in which that variable is used. That is to say, if you assign a string value to variable $var, $var becomes a string. If you then assign an integer value to $var, it becomes an integer.

An example of PHP's automatic type conversion is the addition operator '+'. If any of the operands is a float, then all operands are evaluated as floats, and the result will be a float. Otherwise, the operands will be interpreted as integers, and the result will also be an integer. Note that this does NOT change the types of the operands themselves; the only change is in how the operands are evaluated.

```
<?php
$foo = "0";  // $foo is string (ASCII 48)
<!-- bad example, no real operator (must be used with variable, modifies it too)
$foo++;      // $foo is the string "1" (ASCII 49)
-->
$foo += 2;   // $foo is now an integer (2)
$foo = $foo + 1.3;  // $foo is now a float (3.3)
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs";     // $foo is integer (15)
<!--

TODO: explain ++/- - behaviour with strings

examples:

++'001' = '002'
++'abc' = 'abd'
++'xyz' = 'xza'
++'9.9' = '9.0'
++'-3'  = '-4'
- -'9'   = 8 (integer!)
- -'5.5' = '5.5'
- -'-9'  = -10 (integer)
- -'09'  = 8 (integer)
- -'abc' = 'abc'

-->
?>
```

If the last two examples above seem odd, see String conversion to numbers.

If you wish to force a variable to be evaluated as a certain type, see the section on Type casting. If you wish to change the type of a variable, see **settype()**.

If you would like to test any of the examples in this section, you can use the **var_dump()** function.

> **Note:** The behaviour of an automatic conversion to array is currently undefined.

```
<?php
$a = "1";     // $a is a string
$a[0] = "f";  // What about string offsets? What happens?
?>
```

Since PHP (for historical reasons) supports indexing into strings via offsets using the same syntax as array indexing, the example above leads to a problem: should $a become an array with its first element being "f", or should "f" become the first character of the string $a?

The current versions of PHP interpret the second assignment as a string offset identification, so $a becomes "f",

"f" become the first character of the string $a?

The current versions of PHP interpret the second assignment as a string offset identification, so $a becomes "f", the result of this automatic conversion however should be considered undefined. PHP 4 introduced the new curly bracket syntax to access characters in string, use this syntax instead of the one presented above:

```
<?php
$a    = "abc"; // $a is a string
$a{1} = "f";   // $a is now "afc"
?>
```

See the section titled String access by character for more informaton.

Type Casting

Type casting in PHP works much as it does in C: the name of the desired type is written in parentheses before the variable which is to be cast.

```
<?php
$foo = 10;   // $foo is an integer
$bar = (boolean) $foo;   // $bar is a boolean
?>
```

The casts allowed are:

- (int), (integer) - cast to integer
- (bool), (boolean) - cast to boolean
- (float), (double), (real) - cast to float
- (string) - cast to string
- (array) - cast to array
- (object) - cast to object

Note that tabs and spaces are allowed inside the parentheses, so the following are functionally equivalent:

```
<?php
$foo = (int) $bar;
$foo = ( int ) $bar;
?>
```

Note: Instead of casting a variable to string, you can also enclose the variable in double quotes.

```
<?php
$foo = 10;           // $foo is an integer
$str = "$foo";       // $str is a string
$fst = (string) $foo; // $fst is also a string

// This prints out that "they are the same"
if ($fst === $str) {
    echo "they are the same";
}
?>
```

It may not be obvious exactly what will happen when casting between certain types. For more info, see these sections:

- Converting to boolean
- Converting to integer
- Converting to float
- Converting to string
- Converting to array
- Converting to object
- Converting to resource

Chapter 8. Variables

Basics

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

Variable names follow the same rules as other labels in PHP. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

Note: For our purposes here, a letter is a-z, A-Z, and the ASCII characters from 127 through 255 (0x7f-0xff).

```
<?php
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // outputs "Bob, Joe"

$4site = 'not yet';     // invalid; starts with a number
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';    // valid; 'ä' is ASCII 228.
```

```
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';    // valid; 'ä' is ASCII 228.
?>
```

In PHP 3, variables are always assigned by value. That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. This means, for instance, that after assigning one variable's value to another, changing one of those variables will have no effect on the other. For more information on this kind of assignment, see the chapter on Expressions.

PHP 4 offers another way to assign values to variables: assign by reference. This means that the new variable simply references (in other words, "becomes an alias for" or "points to") the original variable. Changes to the new variable affect the original, and vice versa. This also means that no copying is performed; thus, the assignment happens more quickly. However, any speedup will likely be noticed only in tight loops or when assigning large arrays or objects.

To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable). For instance, the following code snippet outputs 'My name is Bob' twice:

```
<?php
$foo = 'Bob';              // Assign the value 'Bob' to $foo
$bar = &$foo;              // Reference $foo via $bar.
$bar = "My name is $bar";  // Alter $bar...
echo $bar;
echo $foo;                 // $foo is altered too.
?>
```

One important thing to note is that only named variables may be assigned by reference.

```
<?php
$foo = 25;
$bar = &$foo;      // This is a valid assignment.
$bar = &(24 * 7);  // Invalid; references an unnamed expression.

function test()
{
   return 25;
}

$bar = &test();    // Invalid.
?>
```

PHP follows Perl's convention when dealing with arithmetic operations on character variables and not C's. For example, in Perl 'Z'+1 turns into 'AA', while in C 'Z'+1 turns into '[' (ord('Z') == 90, ord('[') == 91).

Example 8-1. Arithmetic Operations on Character Variables

```
<?php
$i = 'W';
for($n=0; $n<6; $n++)
  echo ++$i . "\n";

/*
  Produces the output similar to the following:

X
Y
Z
AA
AB
AC

*/
?>
```

Predefined variables

PHP provides a large number of predefined variables to any script which it runs. Many of these variables, however, cannot be fully documented as they are dependent upon which server is running, the version and setup of the server, and other factors. Some of these variables will not be available when PHP is run on the command line. For a listing of these variables, please see the section on Reserved Predefined Variables.

Warning
In PHP 4.2.0 and later, the default value for the PHP directive register_globals is off. This is a major change in PHP. Having register_globals off affects the set of predefined variables available in the global scope. For example, to get DOCUMENT_ROOT you'll use $_SERVER['DOCUMENT_ROOT'] instead of $DOCUMENT_ROOT, or $_GET['id'] from the URL http://www.example.com/test.php?id=3 instead of $id, or $_ENV['HOME'] instead of $HOME. For related information on this change, read the configuration entry for register_globals, the security chapter on Using Register Globals , as well as the PHP 4.1.0 and 4.2.0 Release Announcements. Using the available PHP Reserved Predefined Variables, like the superglobal arrays, is preferred.

From version 4.1.0 onward, PHP provides an additional set of predefined arrays containing variables from the web server (if applicable), the environment, and user input. These new arrays are rather special in that they are automatically global--i.e., automatically available in every scope. For this reason, they are often known as 'autoglobals' or 'superglobals'. (There is no mechanism in PHP for user-defined superglobals.) The superglobals are listed below: however, for a listing of their contents and further discussion on PHP predefined variables and their natures, please see the section Reserved Predefined Variables. Also, you'll notice how the older predefined variables ($HTTP_*_VARS) still exist.

further discussion on PHP predefined variables and their natures, please see the section Reserved Predefined Variables. Also, you'll notice how the older predefined variables ($HTTP_*_VARS) still exist.

Variable variables: Superglobals cannot be used as variable variables.

If certain variables in variables_order are not set, their appropriate PHP predefined arrays are also left empty.

PHP Superglobals

$GLOBALS

Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables. $GLOBALS has existed since PHP 3.

$_SERVER

Variables set by the web server or otherwise directly related to the execution environment of the current script. Analogous to the old $HTTP_SERVER_VARS array (which is still available, but deprecated).

$_GET

Variables provided to the script via HTTP GET. Analogous to the old $HTTP_GET_VARS array (which is still available, but deprecated).

$_POST

Variables provided to the script via HTTP POST. Analogous to the old $HTTP_POST_VARS array (which is still available, but deprecated).

$_COOKIE

Variables provided to the script via HTTP cookies. Analogous to the old $HTTP_COOKIE_VARS array (which is still available, but deprecated).

$_FILES

Variables provided to the script via HTTP post file uploads. Analogous to the old $HTTP_POST_FILES array (which is still available, but deprecated). See POST method uploads for more information.

$_ENV

Variables provided to the script via the environment. Analogous to the old $HTTP_ENV_VARS array (which is still available, but deprecated).

$_REQUEST

Variables provided to the script via any user input mechanism, and which therefore cannot be trusted. The presence and order of variable inclusion in this array is defined according to the variables_order configuration directive. This array has no direct analogue in versions of PHP prior to 4.1.0. See also i**mport_request_variables()**.

Note: When running on the command line , this will not include the argv and argc entries; these are present in the $_SERVER array.

$_SESSION

Variables which are currently registered to a script's session. Analogous to the old $HTTP_SESSION_VARS array (which is still available, but deprecated). See the Session handling functions section for more information.

Variable scope

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. This single scope spans included and required files as well. For example:

```
<?php
$a = 1;
include "b.inc";
?>
```

Here the $a variable will be available within the included b.inc script. However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
<?php
$a = 1; /* global scope */

function Test()
{
    echo $a; /* reference to local scope variable */
}

Test();
?>
```

This script will not produce any output because the echo statement refers to a local version of the $a variable, and it has not been assigned a value within this scope. You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition. This can cause some problems in that people may inadvertently change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function. An example:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
    global $a, $b;

    $b = $a + $b;
}
```

```
    $b = $a + $b;
}

Sum();
echo $b;
?>
```

The above script will output "3". By declaring $a and $b global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

A second way to access variables from the global scope is to use the special PHP-defined $GLOBALS array. The previous example can be rewritten as:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum();
echo $b;
?>
```

The $GLOBALS array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element. Notice how $GLOBALS exists in any scope, this is because $GLOBALS is a superglobal. Here's an example demonstrating the power of superglobals:

```
<?php
function test_global()
{
    // Most predefined variables aren't "super" and require
    // 'global' to be available to the functions local scope.
    global $HTTP_POST_VARS;

    print $HTTP_POST_VARS['name'];

    // Superglobals are available in any scope and do
    // not require 'global'.  Superglobals are available
    // as of PHP 4.1.0
    print $_POST['name'];
}
?>
```

Another important feature of variable scoping is the static variable. A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope. Consider the following example:

```
<?php
function Test ()
{
    $a = 0;
    echo $a;
    $a++;
}
?>
```

This function is quite useless since every time it is called it sets $a to 0 and prints "0". The $a++ which increments the variable serves no purpose since as soon as the function exits the $a variable disappears. To make a useful counting function which will not lose track of the current count, the $a variable is declared static:

```
<?php
function Test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Now, every time the Test() function is called it will print the value of $a and increment it.

Static variables also provide one way to deal with recursive functions. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it recurse indefinitely. You must make sure you have an adequate way of terminating the recursion. The following simple function recursively counts to 10, using the static variable $count to know when to stop:

```
<?php
function Test()
{
    static $count = 0;

    $count++;
```

```
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
?>
```

The Zend Engine 1, driving PHP4, implements the static and global modifier for variables in terms of references. For example, a true global variable imported inside a function scope with the global statement actually creates a reference to the global variable. This can lead to unexpected behaviour which the following example addresses:

```
<?php
function test_global_ref() {
    global $obj;
    $obj = &new stdclass;
}

function test_global_noref() {
    global $obj;
    $obj = new stdclass;
}

test_global_ref();
var_dump($obj);
test_global_noref();
var_dump($obj);
?>
```

Executing this example will result in the following output:

```
NULL
object(stdClass)(0) {
}
```

A similar behaviour applies to the static statement. References are not stored statically:

```
<?php
function &get_instance_ref() {
    static $obj;

    echo "Static object: ";
    var_dump($obj);
    if (!isset($obj)) {
        // Assign a reference to the static variable
        $obj = &new stdclass;
    }
    $obj->property++;
    return $obj;
}

function &get_instance_noref() {
    static $obj;

    echo "Static object: ";
    var_dump($obj);
    if (!isset($obj)) {
        // Assign the object to the static variable
        $obj = new stdclass;
    }
    $obj->property++;
    return $obj;
}

$obj1 = get_instance_ref();
$still_obj1 = get_instance_ref();
echo "\n";
$obj2 = get_instance_noref();
$still_obj2 = get_instance_noref();
?>
```

Executing this example will result in the following output:

```
Static object: NULL
Static object: NULL

Static object: NULL
Static object: object(stdClass)(1) {
  ["property"]=>
  int(1)
}
```

This example demonstrates that when assigning a reference to a static variable, it's not remembered when you call the &get_instance_ref() function a second time.

Variable variables

Variable variables

Sometimes it is convenient to be able to have variable variable names. That is, a variable name which can be set and used dynamically. A normal variable is set with a statement such as:

```
<?php
$a = "hello";
?>
```

A variable variable takes the value of a variable and treats that as the name of a variable. In the above example, hello, can be used as the name of a variable by using two dollar signs. i.e.

```
<?php
$$a = "world";
?>
```

At this point two variables have been defined and stored in the PHP symbol tree: $a with contents "hello" and $hello with contents "world". Therefore, this statement:

```
<?php
echo "$a ${$a}";
?>
```

produces the exact same output as:

```
<?php
echo "$a $hello";
?>
```

i.e. they both produce: hello world.

In order to use variable variables with arrays, you have to resolve an ambiguity problem. That is, if you write $$a[1] then the parser needs to know if you meant to use $a[1] as a variable, or if you wanted $$a as the variable and then the [1] index from that variable. The syntax for resolving this ambiguity is: ${$a[1]} for the first case and ${$a}[1] for the second.

Warning
Please note that variable variables cannot be used with PHP's Superglobal arrays. This means you cannot do things like ${$_GET}. If you are looking for a way to handle availability of superglobals and the old HTTP_*_VARS, you might want to try referencing them.

Variables from outside PHP

HTML Forms (GET and POST)

When a form is submitted to a PHP script, the information from that form is automatically made available to the script. There are many ways to access this information, for example:

Example 8-2. A simple HTML form

```
<form action="foo.php" method="post">
    Name:  <input type="text" name="username"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" name="submit" value="Submit me!">
</form>
```

Depending on your particular setup and personal preferences, there are many ways to access data from your HTML forms. Some examples are:

Example 8-3. Accessing data from a simple POST HTML form

```
<?php
// Available since PHP 4.1.0

   print $_POST['username'];
   print $_REQUEST['username'];

   import_request_variables('p', 'p_');
   print $p_username;

// Available since PHP 3.

   print $HTTP_POST_VARS['username'];

// Available if the PHP directive register_globals = on.  As of
// PHP 4.2.0 the default value of register_globals = off.
// Using/relying on this method is not preferred.

   print $username;
?>
```

```
    print $username;
?>
```

Using a GET form is similar except you'll use the appropriate GET predefined variable instead. GET also applies to the QUERY_STRING (the information after the '?' in an URL). So, for example, http://www.example.com/test.php?id=3 contains GET data which is accessible with $_GET['id']. See also $_REQUEST and **import_request_variables()**.

> **Note:** Superglobal arrays, like $_POST and $_GET, became available in PHP 4.1.0

As shown, before PHP 4.2.0 the default value for register_globals was on. And, in PHP 3 it was always on. The PHP community is encouraging all to not rely on this directive as it's preferred to assume it's off and code accordingly.

> **Note:** The magic_quotes_gpc configuration directive affects Get, Post and Cookie values. If turned on, value (It's "PHP!") will automagically become (It\'s \"PHP!\"). Escaping is needed for DB insertion. See also **addslashes()**, **stripslashes()** and magic_quotes_sybase.

PHP also understands arrays in the context of form variables (see the related faq). You may, for example, group related variables together, or use this feature to retrieve values from a multiple select input. For example, let's post a form to itself and upon submission display the data:

Example 8-4. More complex form variables

```
<?php
if ($HTTP_POST_VARS['action'] == 'submitted') {
    print '<pre>';

    print_r($HTTP_POST_VARS);
    print '<a href="'. $HTTP_SERVER_VARS['PHP_SELF'] .'">Please try again</a>';

    print '</pre>';
} else {
?>
<form action="<?php echo $HTTP_SERVER_VARS['PHP_SELF']; ?>" method="post">
    Name:  <input type="text" name="personal[name]"><br>
    Email: <input type="text" name="personal[email]"><br>
    Beer: <br>
    <select multiple name="beer[]">
        <option value="warthog">Warthog</option>
        <option value="guinness">Guinness</option>
        <option value="stuttgarter">Stuttgarter Schwabenbräu</option>
    </select><br>
    <input type="hidden" name="action" value="submitted">
    <input type="submit" name="submit" value="submit me!">
</form>
<?php
}
?>
```

In PHP 3, the array form variable usage is limited to single-dimensional arrays. In PHP 4, no such restriction applies.

IMAGE SUBMIT variable names

When submitting a form, it is possible to use an image instead of the standard submit button with a tag like:

```
<input type="image" src="image.gif" name="sub">
```

When the user clicks somewhere on the image, the accompanying form will be transmitted to the server with two additional variables, sub_x and sub_y. These contain the coordinates of the user click within the image. The experienced may note that the actual variable names sent by the browser contains a period rather than an underscore, but PHP converts the period to an underscore automatically.

HTTP Cookies

PHP transparently supports HTTP cookies as defined by Netscape's Spec. Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the **setcookie()** function. Cookies are part of the HTTP header, so the SetCookie function must be called before any output is sent to the browser. This is the same restriction as for the **header()** function. Cookie data is then available in the appropriate cookie data arrays, such as $_COOKIE, $HTTP_COOKIE_VARS as well as in $_REQUEST. See the **setcookie()** manual page for more details and examples.

If you wish to assign multiple values to a single cookie variable, you may assign it as an array. For example:

```
<?php
  setcookie("MyCookie[foo]", "Testing 1", time()+3600);
  setcookie("MyCookie[bar]", "Testing 2", time()+3600);
?>
```

That will create two seperate cookies although MyCookie will now be a single array in your script. If you want to set just one cookie with multiple values, consider using **serialize()** or **explode()** on the value first.

Note that a cookie will replace a previous cookie by the same name in your browser unless the path or domain is different. So, for a shopping cart application you may want to keep a counter and pass this along. i.e.

Example 8-5. A setcookie() example

```
<?php
$count++;
setcookie("count", $count, time()+3600);
setcookie("Cart[$count]", $item, time()+3600);
```

```
$count++;
setcookie("count", $count, time()+3600);
setcookie("Cart[$count]", $item, time()+3600);
?>
```

Dots in incoming variable names

Typically, PHP does not alter the names of variables when they are passed into a script. However, it should be noted that the dot (period, full stop) is not a valid character in a PHP variable name. For the reason, look at it:

```
<?php
$varname.ext;  /* invalid variable name */
?>
```

Now, what the parser sees is a variable named $varname, followed by the string concatenation operator, followed by the barestring (i.e. unquoted string which doesn't match any known key or reserved words) 'ext'. Obviously, this doesn't have the intended result.

For this reason, it is important to note that PHP will automatically replace any dots in incoming variable names with underscores.

Determining variable types

Because PHP determines the types of variables and converts them (generally) as needed, it is not always obvious what type a given variable is at any one time. PHP includes several functions which find out what type a variable is, such as: **gettype()**, i**s_array()**, i**s_float()**, i**s_int()**, i**s_object()**, and i**s_string()**. See also the chapter on Types.

Chapter 9. Constants

A constant is a identifier (name) for a simple value. As the name suggests, that value cannot change during the execution of the script (except the magic constants which aren't actually constants). A constant is case-sensitive by default. By convention constant identifiers are always uppercase.

The name of a constant follows the same rules as any label in PHP. A valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*

> **Note:** For our purposes here, a letter is a-z, A-Z, and the ASCII characters from 127 through 255 (0x7f-0xff).

The scope of a constant is global--you can access it anywhere in your script without regard to scope.

Syntax

You can define a constant by using the **define()**-function. Once a constant is defined, it can never be changed or undefined.

Only scalar data (**boolean**, i**nteger**, **float** and **string**) can be contained in constants.

You can get the value of a constant by simply specifying its name. Unlike with variables, you should not prepend a constant with a $. You can also use the function **constant()**, to read a constant's value, if you are to obtain the constant's name dynamically. Use **get_defined_constants()** to get a list of all defined constants.

> **Note:** Constants and (global) variables are in a different namespace. This implies that for example **TRUE** and $TRUE are generally different.

If you use an undefined constant, PHP assumes that you mean the name of the constant itself. A notice will be issued when this happens. Use the **defined()**-function if you want to know if a constant is set.

These are the differences between constants and variables:

- Constants do not have a dollar sign ($) before them;
- Constants may only be defined using the **define()** function, not by simple assignment;
- Constants may be defined and accessed anywhere without regard to variable scoping rules;
- Constants may not be redefined or undefined once they have been set; and
- Constants may only evaluate to scalar values.

Example 9-1. Defining Constants

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.
?>
```

Predefined constants

PHP provides a large number of predefined constants to any script which it runs. Many of these constants, however, are created by various extensions, and will only be present when those extensions are available, either via dynamic loading or because they have been compiled in.

There are four magical constants that change depending on where they're used. For example, the value of **__LINE__** depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

Table 9-1. A few "magical" PHP "constants"

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file.
__FUNCTION__	The function name. This was added in PHP 4.3.0
__CLASS__	The class name. This was added in PHP 4.3.0

__CLASS__	The class name. This was added in PHP 4.3.0

A list of predefined constants is available in the section Reserved predefined constants.

Chapter 10. Expressions

Expressions are the most important building stones of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".

The most basic forms of expressions are constants and variables. When you type "$a = 5", you're assigning '5' into $a. '5', obviously, has the value 5, or in other words '5' is an expression with the value of 5 (in this case, '5' is an integer constant).

After this assignment, you'd expect $a's value to be 5 as well, so if you wrote $b = $a, you'd expect it to behave just as if you wrote $b = 5. In other words, $a is an expression with the value of 5 as well. If everything works right, this is exactly what will happen.

Slightly more complex examples for expressions are functions. For instance, consider the following function:

```
function foo ()
{
    return 5;
}
```

Assuming you're familiar with the concept of functions (if you're not, take a look at the chapter about functions), you'd assume that typing $c = foo() is essentially just like writing $c = 5, and you're right. Functions are expressions with the value of their return value. Since foo() returns 5, the value of the expression 'foo()' is 5. Usually functions don't just return a static value but compute something.

Of course, values in PHP don't have to be integers, and very often they aren't. PHP supports three scalar value types: integer values, floating point values and string values (scalar values are values that you can't 'break' into smaller pieces, unlike arrays, for instance). PHP also supports two composite (non-scalar) types: arrays and objects. Each of these value types can be assigned into variables or returned from functions.

So far, users of PHP/FI 2 shouldn't feel any change. However, PHP takes expressions much further, in the same way many other languages do. PHP is an expression-oriented language, in the sense that almost everything is an expression. Consider the example we've already dealt with, '$a = 5'. It's easy to see that there are two values involved here, the value of the integer constant '5', and the value of $a which is being updated to 5 as well. But the truth is that there's one additional value involved here, and that's the value of the assignment itself. The assignment itself evaluates to the assigned value, in this case 5. In practice, it means that '$a = 5', regardless of what it does, is an expression with the value 5. Thus, writing something like '$b = ($a = 5)' is like writing '$a = 5; $b = 5;' (a semicolon marks the end of a statement). Since assignments are parsed in a right to left order, you can also write '$b = $a = 5'.

Another good example of expression orientation is pre- and post-increment and decrement. Users of PHP/FI 2 and many other languages may be familiar with the notation of variable++ and variable--. These are increment and decrement operators. In PHP/FI 2, the statement '$a++' has no value (is not an expression), and thus you can't assign it or use it in any way. PHP enhances the increment/decrement capabilities by making these expressions as well, like in C. In PHP, like in C, there are two types of increment - pre-increment and post-increment. Both pre-increment and post-increment essentially increment the variable, and the effect on the variable is idential. The difference is with the value of the increment expression. Pre-increment, which is written '++$variable', evaluates to the incremented value (PHP increments the variable before reading its value, thus the name 'pre-increment'). Post-increment, which is written '$variable++' evaluates to the original value of $variable, before it was incremented (PHP increments the variable after reading its value, thus the name 'post-increment').

A very common type of expressions are comparison expressions. These expressions evaluate to either 0 or 1, meaning **FALSE** or **TRUE** (respectively). PHP supports > (bigger than), >= (bigger than or equal to), == (equal), != (not equal), < (smaller than) and <= (smaller than or equal to). These expressions are most commonly used inside conditional execution, such as if statements.

The last example of expressions we'll deal with here is combined operator-assignment expressions. You already know that if you want to increment $a by 1, you can simply write '$a++' or '++$a'. But what if you want to add more than one to it, for instance 3? You could write '$a++' multiple times, but this is obviously not a very efficient or comfortable way. A much more common practice is to write '$a = $a + 3'. '$a + 3' evaluates to the value of $a plus 3, and is assigned back into $a, which results in incrementing $a by 3. In PHP, as in several other languages like C, you can write this in a shorter way, which with time would become clearer and quicker to understand as well. Adding 3 to the current value of $a can be written '$a += 3'. This means exactly "take the value of $a, add 3 to it, and assign it back into $a". In addition to being shorter and clearer, this also results in faster execution. The value of '$a += 3', like the value of a regular assignment, is the assigned value. Notice that it is NOT 3, but the combined value of $a plus 3 (this is the value that's assigned into $a). Any two-place operator can be used in this operator-assignment mode, for example '$a -= 5' (subtract 5 from the value of $a), '$b *= 7' (multiply the value of $b by 7), etc.

There is one more expression that may seem odd if you haven't seen it in other languages, the ternary conditional operator:

```
$first ? $second : $third
```

If the value of the first subexpression is **TRUE** (non-zero), then the second subexpression is evaluated, and that is the result of the conditional expression. Otherwise, the third subexpression is evaluated, and that is the value.

The following example should help you understand pre- and post-increment and expressions in general a bit better:

```
function double($i)
{
    return $i*2;
}
$b = $a = 5;        /* assign the value five into the variable $a and $b */
$c = $a++;          /* post-increment, assign original value of $a
                    (5) to $c */
$e = $d = ++$b;     /* pre-increment, assign the incremented value of
                    $b (6) to $d and $e */

/* at this point, both $d and $e are equal to 6 */

$f = double($d++);  /* assign twice the value of $d <emphasis>before</emphasis>
                    the increment, 2*6 = 12 to $f */
$g = double(++$e);  /* assign twice the value of $e <emphasis>after</emphasis>
```

```
                the increment, 2*6 = 12 to $f */
$g = double(++$e);  /* assign twice the value of $e <emphasis>after</emphasis>
                the increment, 2*7 = 14 to $g */
$h = $g += 10;      /* first, $g is incremented by 10 and ends with the
                value of 24. the value of the assignment (24) is
                then assigned into $h, and $h ends with the value
                of 24 as well. */
```

In the beginning of the chapter we said that we'll be describing the various statement types, and as promised, expressions can be statements. However, not every expression is a statement. In this case, a statement has the form of 'expr' ';' that is, an expression followed by a semicolon. In '$b=$a=5;', $a=5 is a valid expression, but it's not a statement by itself. '$b=$a=5;' however is a valid statement.

One last thing worth mentioning is the truth value of expressions. In many events, mainly in conditional execution and loops, you're not interested in the specific value of the expression, but only care about whether it means **TRUE** or **FALSE**. The constants **TRUE** and **FALSE** (case-insensitive) are the two possible boolean values. When necessary, an expression is automatically converted to boolean. See the section about type-casting for details about how.

PHP provides a full and powerful implementation of expressions, and documenting it entirely goes beyond the scope of this manual. The above examples should give you a good idea about what expressions are and how you can construct useful expressions. Throughout the rest of this manual we'll write expr to indicate any valid PHP expression.

Chapter 11. Operators

Operator Precedence

The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression 1 + 5 * 3, the answer is 16 and not 18 because the multiplication ("*") operator has a higher precedence than the addition ("+") operator. Parentheses may be used to force precedence, if necessary. For instance: (1 + 5) * 3 evaluates to 18.

The following table lists the precedence of operators with the lowest-precedence operators listed first.

Table 11-1. Operator Precedence

Associativity	Operators
left	,
left	or
left	xor
left	and
right	print
left	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
left	? :
left	||
left	&&
left	|
left	^
left	&
non-associative	== != === !==
non-associative	< <= > >=
left	<< >>
left	+ - .
left	* / %
right	! ~ ++ -- (int) (float) (string) (array) (object) @
right	[
non-associative	new

> **Note:** Although ! has a higher precedence than =, PHP will still allow expressions similar to the following: if (!$a = foo()), in which case the output from foo() is put into $a.

Arithmetic Operators

Remember basic arithmetic from school? These work just like those.

Table 11-2. Arithmetic Operators

Example	Name	Result
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulus	Remainder of $a divided by $b.

The division operator ("/") returns a float value anytime, even if the two operands are integers (or strings that get converted to integers).

Assignment Operators

The basic assignment operator is "=". Your first inclination might be to think of this as "equal to". Don't. It really means that the the left operand gets set to the value of the expression on the rights (that is, "gets set to").

The value of an assignment expression is the value assigned. That is, the value of "$a = 3" is 3. This allows you to do some tricky things:

The value of an assignment expression is the value assigned. That is, the value of "$a = 3" is 3. This allows you to do some tricky things:

```
$a = ($b = 4) + 5; // $a is equal to 9 now, and $b has been set to 4.
```

In addition to the basic assignment operator, there are "combined operators" for all of the binary arithmetic and string operators that allow you to use a value in an expression and then set its value to the result of that expression. For example:

```
$a = 3;
$a += 5; // sets $a to 8, as if we had said: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // sets $b to "Hello There!", just like $b = $b . "There!";
```

Note that the assignment copies the original variable to the new one (assignment by value), so changes to one will not affect the other. This may also have relevance if you need to copy something like a large array inside a tight loop. PHP 4 supports assignment by reference, using the $var = &$othervar; syntax, but this is not possible in PHP 3. 'Assignment by reference' means that both variables end up pointing at the same data, and nothing is copied anywhere. To learn more about references, please read References explained.

Bitwise Operators

Bitwise operators allow you to turn specific bits within an integer on or off. If both the left- and right-hand parameters are strings, the bitwise operator will operate on the characters in this string.

```
<?php
    echo 12 ^ 9; // Outputs '5'

    echo "12" ^ "9"; // Outputs the Backspace character (ascii 8)
                     // ('1' (ascii 49)) ^ ('9' (ascii 57)) = #8

    echo "hallo" ^ "hello"; // Outputs the ascii values #0 #4 #0 #0 #0
                            // 'a' ^ 'e' = #4
?>
```

Table 11-3. Bitwise Operators

Example	Name	Result
$a & $b	And	Bits that are set in both $a and $b are set.
$a | $b	Or	Bits that are set in either $a or $b are set.
$a ^ $b	Xor	Bits that are set in $a or $b but not both are set.
~ $a	Not	Bits that are set in $a are not set, and vice versa.
$a << $b	Shift left	Shift the bits of $a $b steps to the left (each step means "multiply by two")
$a >> $b	Shift right	Shift the bits of $a $b steps to the right (each step means "divide by two")

Comparison Operators

Comparison operators, as their name implies, allow you to compare two values.

Table 11-4. Comparison Operators

Example	Name	Result
$a == $b	Equal	**TRUE** if $a is equal to $b.
$a === $b	Identical	**TRUE** if $a is equal to $b, and they are of the same type. (PHP 4 only)
$a != $b	Not equal	**TRUE** if $a is not equal to $b.
$a <> $b	Not equal	**TRUE** if $a is not equal to $b.
$a !== $b	Not identical	**TRUE** if $a is not equal to $b, or they are not of the same type. (PHP 4 only)
$a < $b	Less than	**TRUE** if $a is strictly less than $b.
$a > $b	Greater than	**TRUE** if $a is strictly greater than $b.
$a <= $b	Less than or equal to	**TRUE** if $a is less than or equal to $b.
$a >= $b	Greater than or equal to	**TRUE** if $a is greater than or equal to $b.

Another conditional operator is the "?:" (or ternary) operator, which operates as in C and many other languages.

```
(expr1) ? (expr2) : (expr3);
```

This expression evaluates to expr2 if expr1 evaluates to **TRUE**, and expr3 if expr1 evaluates to **FALSE**.

Error Control Operators

PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.

If the track_errors feature is enabled, any error message generated by the expression will be saved in the variable $php_errormsg. This variable will be overwritten on each error, so check early if you want to use it.

```
<?php
/* Intentional file error */
$my_file = @file ('non_existent_file') or
    die ("Failed opening file: error was '$php_errormsg'");

// this works for any expression, not just functions:
$value = @$cache[$key];
// will not issue a notice if the index $key doesn't exist.

?>
```

> **Note:** The @-operator works only on expressions. A simple rule of thumb is: if you can take the value of something, you can prepend the @ operator to it. For instance, you can prepend it to variables, function and **include()** calls, constants, and so forth. You cannot prepend it to function or class definitions, or conditional structures such as if and foreach, and so forth.

See also **error_reporting()**.

> **Note:** The "@" error-control operator prefix will not disable messages that are the result of parse errors.

Warning
Currently the "@" error-control operator prefix will even disable error reporting for critical errors that will terminate script execution. Among other things, this means that if you use "@" to suppress errors from a certain function and either it isn't available or has been mistyped, the script will die right there with no indication as to why.

Execution Operators

PHP supports one execution operator: backticks (``). Note that these are not single-quotes! PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won't simply be dumped to output; it can be assigned to a variable).

```
$output = `ls -al`;
echo "<pre>$output</pre>";
```

> **Note:** The backtick operator is disabled when safe mode is enabled or **shell_exec()** is disabled.

See also **escapeshellcmd()**, **exec()**, **passthru()**, **popen()**, **shell_exec()**, and **system()**.

Incrementing/Decrementing Operators

PHP supports C-style pre- and post-increment and decrement operators.

Table 11-5. Increment/decrement Operators

Example	Name	Effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
--$a	Pre-decrement	Decrements $a by one, then returns $a.
$a--	Post-decrement	Returns $a, then decrements $a by one.

Here's a simple example script:

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Should be 5: " . $a++ . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "Should be 6: " . ++$a . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";

echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Should be 5: " . $a-- . "<br />\n";
echo "Should be 4: " . $a . "<br />\n";

echo "<h3>Predecrement</h3>";
$a = 5;
echo "Should be 4: " . --$a . "<br />\n";
echo "Should be 4: " . $a . "<br />\n";
?>
```

Logical Operators

Table 11-6. Logical Operators

Example	Name	Result
$a and $b	And	**TRUE** if both $a and $b are **TRUE**.
$a or $b	Or	**TRUE** if either $a or $b is **TRUE**.

$a and $b	And	**TRUE** if both $a and $b are **TRUE**.
$a or $b	Or	**TRUE** if either $a or $b is **TRUE**.
$a xor $b	Xor	**TRUE** if either $a or $b is **TRUE**, but not both.
! $a	Not	**TRUE** if $a is not **TRUE**.
$a && $b	And	**TRUE** if both $a and $b are **TRUE**.
$a || $b	Or	**TRUE** if either $a or $b is **TRUE**.

The reason for the two different variations of "and" and "or" operators is that they operate at different precedences. (See Operator Precedence.)

String Operators

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side. Please read Assignment Operators for more information.

```
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!";     // now $a contains "Hello World!"
```

Array Operators

The only array operator in PHP is the + operator. It appends the right handed array to the left handed, whereas duplicated keys are NOT overwritten.

```
$a = array("a" => "apple", "b" => "banana");
$b = array("a" =>"pear", "b" => "strawberry", "c" => "cherry");

$c = $a + $b;

var_dump($c);
```

```
array(3) {
  ["a"]=>
  string(5) "apple"
  ["b"]=>
  string(6) "banana"
  ["c"]=>
  string(6) "cherry"
}
```

Chapter 12. Control Structures

Any PHP script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement of even a statement that does nothing (an empty statement). Statements usually end with a semicolon. In addition, statements can be grouped into a statement-group by encapsulating a group of statements with curly braces. A statement-group is a statement by itself as well. The various statement types are described in this chapter.

---

if

The if construct is one of the most important features of many languages, PHP included. It allows for conditional execution of code fragments. PHP features an if structure that is similar to that of C:

```
if (expr)
    statement
```

As described in the section about expressions, expr is evaluated to its Boolean value. If expr evaluates to **TRUE**, PHP will execute statement, and if it evaluates to **FALSE** - it'll ignore it. More information about what values evaluate to **FALSE** can be found in the 'Converting to boolean' section.

The following example would display a is bigger than b if $a is bigger than $b:

```
if ($a > $b)
    print "a is bigger than b";
```

Often you'd want to have more than one statement to be executed conditionally. Of course, there's no need to wrap each statement with an if clause. Instead, you can group several statements into a statement group. For example, this code would display a is bigger than b if $a is bigger than $b, and would then assign the value of $a into $b:

```
if ($a > $b) {
    print "a is bigger than b";
    $b = $a;
}
```

If statements can be nested indefinitely within other if statements, which provides you with complete flexibility for conditional execution of the various parts of your program.

---

else

Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is what else is for. else extends an if statement to execute a statement in case the expression in the if statement evaluates to **FALSE**. For example, the following code would display a is bigger than b if $a is bigger than $b, and a is NOT bigger than b otherwise:

```
if ($a > $b) {
    print "a is bigger than b";
} else {
    print "a is NOT bigger than b";
}
```

The else statement is only executed if the if expression evaluated to **FALSE**, and if there were any elseif expressions - only if they evaluated to **FALSE** as well (see elseif).

---

elseif

elseif, as its name suggests, is a combination of if and else. Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to **FALSE**. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to **TRUE**. For example, the following code would display a is bigger than b, a equal to b or a is smaller than b:

```
if ($a > $b) {
    print "a is bigger than b";
} elseif ($a == $b) {
    print "a is equal to b";
} else {
    print "a is smaller than b";
}
```

There may be several elseifs within the same if statement. The first elseif expression (if any) that evaluates to **TRUE** would be executed. In PHP, you can also write 'else if' (in two words) and the behavior would be identical to the one of 'elseif' (in a single word). The syntactic meaning is slightly different (if you're familiar with C, this is the same behavior) but the bottom line is that both would result in exactly the same behavior.

The elseif statement is only executed if the preceding if expression and any preceding elseif expressions evaluated to **FALSE**, and the current elseif expression evaluated to **TRUE**.

Alternative syntax for control structures

PHP offers an alternative syntax for some of its control structures; namely, if, while, for, foreach, and switch. In each case, the basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to endif;, endwhile;, endfor;, endforeach;, or endswitch;, respectively.

```
<?php if ($a == 5): ?>
A is equal to 5
<?php endif; ?>
```

In the above example, the HTML block "A is equal to 5" is nested within an if statement written in the alternative syntax. The HTML block would be displayed only if $a is equal to 5.

The alternative syntax applies to else and elseif as well. The following is an if structure with elseif and else in the alternative format:

```
if ($a == 5):
    print "a equals 5";
    print "...";
elseif ($a == 6):
    print "a equals 6";
    print "!!!";
else:
    print "a is neither 5 nor 6";
endif;
```

See also while, for, and if for further examples.

---

while

while loops are the simplest type of loop in PHP. They behave just like their C counterparts. The basic form of a while statement is:

```
while (expr) statement
```

The meaning of a while statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the while expression evaluates to **TRUE**. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP

The meaning of a while statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the while expression evaluates to **TRUE**. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration). Sometimes, if the while expression evaluates to **FALSE** from the very beginning, the nested statement(s) won't even be run once.

Like with the if statement, you can group multiple statements within the same while loop by surrounding a group of statements with curly braces, or by using the alternate syntax:

```
while (expr): statement ... endwhile;
```

The following examples are identical, and both print numbers from 1 to 10:

```
/* example 1 */

$i = 1;
while ($i <= 10) {
    print $i++;  /* the printed value would be
                    $i before the increment
                    (post-increment) */
}

/* example 2 */

$i = 1;
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

---

do..while

do..while loops are very similar to while loops, except the truth expression is checked at the end of each iteration instead of in the beginning. The main difference from regular while loops is that the first iteration of a do..while loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it's may not necessarily run with a regular while loop (the truth expression is checked at the beginning of each iteration, if it evaluates to **FALSE** right from the beginning, the loop execution would end immediately).

There is just one syntax for do..while loops:

```
$i = 0;
do {
   print $i;
} while ($i>0);
```

The above loop would run one time exactly, since after the first iteration, when truth expression is checked, it evaluates to **FALSE** ($i is not bigger than 0) and the loop execution ends.

Advanced C users may be familiar with a different usage of the do..while loop, to allow stopping execution in the middle of code blocks, by encapsulating them with do..while(0), and using the break statement. The following code fragment demonstrates this:

```
do {
    if ($i < 5) {
        print "i is not big enough";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
    print "i is ok";

     ...process i...

} while(0);
```

Don't worry if you don't understand this right away or at all. You can code scripts and even powerful scripts without using this 'feature'.

---

for

for loops are the most complex loops in PHP. They behave like their C counterparts. The syntax of a for loop is:

```
for (expr1; expr2; expr3) statement
```

The first expression (expr1) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, expr2 is evaluated. If it evaluates to **TRUE**, the loop continues and the nested statement(s) are executed. If it evaluates to **FALSE**, the execution of the loop ends.

At the end of each iteration, expr3 is evaluated (executed).

Each of the expressions can be empty. expr2 being empty means the loop should be run indefinitely (PHP implicitly considers it as **TRUE**, like C). This may not be as useless as you might think, since often you'd want to end the loop using a conditional break statement instead of using the for truth expression.

Each of the expressions can be empty. expr2 being empty means the loop should be run indefinitely (PHP implicitly considers it as **TRUE**, like C). This may not be as useless as you might think, since often you'd want to end the loop using a conditional break statement instead of using the for truth expression.

Consider the following examples. All of them display numbers from 1 to 10:

```
/* example 1 */

for ($i = 1; $i <= 10; $i++) {
    print $i;
}

/* example 2 */

for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
}

/* example 3 */

$i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
    $i++;
}

/* example 4 */

for ($i = 1; $i <= 10; print $i, $i++);
```

Of course, the first example appears to be the nicest one (or perhaps the fourth), but you may find that being able to use empty expressions in for loops comes in handy in many occasions.

PHP also supports the alternate "colon syntax" for for loops.

```
for (expr1; expr2; expr3): statement; ...; endfor;
```

Other languages have a foreach statement to traverse an array or hash. PHP 3 has no such construct; PHP 4 does (see foreach). In PHP 3, you can combine while with the **list()** and **each()** functions to achieve the same effect. See the documentation for these functions for an example.

---

foreach

PHP 4 (not PHP 3) includes a foreach construct, much like Perl and some other languages. This simply gives an easy way to iterate over arrays. foreach works only on arrays, and will issue an error when you try to use it on a variable with a different data type or an uninitialized variables. There are two syntaxes; the second is a minor but useful extension of the first:

```
foreach(array_expression as $value) statement
foreach(array_expression as $key => $value) statement
```

The first form loops over the array given by array_expression. On each loop, the value of the current element is assigned to $value and the internal array pointer is advanced by one (so on the next loop, you'll be looking at the next element).

The second form does the same thing, except that the current element's key will be assigned to the variable $key on each loop.

> **Note:** When foreach first starts executing, the internal array pointer is automatically reset to the first element of the array. This means that you do not need to call **reset()** before a foreach loop.

> **Note:** Also note that foreach operates on a copy of the specified array, not the array itself, therefore the array pointer is not modified as with the **each()** construct and changes to the array element returned are not reflected in the original array. However, the internal pointer of the original array is advanced with the processing of the array. Assuming the foreach loop runs to completion, the array's internal pointer will be at the end of the array.

> **Note:** foreach does not support the ability to suppress error messages using '@'.

You may have noticed that the following are functionally identical:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
    echo "Value: $value<br>\n";
}

foreach ($arr as $value) {
    echo "Value: $value<br>\n";
}
```

The following are also functionally identical:

```
reset ($arr);
```

```
reset ($arr);
while (list($key, $value) = each ($arr)) {
    echo "Key: $key; Value: $value<br>\n";
}

foreach ($arr as $key => $value) {
    echo "Key: $key; Value: $value<br>\n";
}
```

Some more examples to demonstrate usages:

```
/* foreach example 1: value only */

$a = array (1, 2, 3, 17);

foreach ($a as $v) {
   print "Current value of \$a: $v.\n";
}

/* foreach example 2: value (with key printed for illustration) */

$a = array (1, 2, 3, 17);

$i = 0; /* for illustrative purposes only */

foreach($a as $v) {
    print "\$a[$i] => $v.\n";
    $i++;
}

/* foreach example 3: key and value */

$a = array (
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);

foreach($a as $k => $v) {
    print "\$a[$k] => $v.\n";
}

/* foreach example 4: multi-dimensional arrays */

$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
$a[1][1] = "z";

foreach($a as $v1) {
    foreach ($v1 as $v2) {
        print "$v2\n";
    }
}

/* foreach example 5: dynamic arrays */

foreach(array(1, 2, 3, 4, 5) as $v) {
    print "$v\n";
}
```

---

break

break ends execution of the current for, foreach while, do..while or switch structure.

break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.

```
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
    if ($val == 'stop') {
        break;    /* You could also write 'break 1;' here. */
    }
    echo "$val<br>\n";
}

/* Using the optional argument. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* Exit only the switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2;  /* Exit the switch and the while. */
    default:
```

```
        break 2;  /* Exit the switch and the while. */
    default:
        break;
    }
}
```

---

continue

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the beginning of the next iteration.

continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

```
while (list ($key, $value) = each ($arr)) {
    if (!($key % 2)) { // skip odd members
        continue;
    }
    do_something_odd ($value);
}

$i = 0;
while ($i++ < 5) {
    echo "Outer<br>\n";
    while (1) {
        echo "  Middle<br>\n";
        while (1) {
            echo "  Inner<br>\n";
            continue 3;
        }
        echo "This never gets output.<br>\n";
    }
    echo "Neither does this.<br>\n";
}
```

---

switch

The switch statement is similar to a series of IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the switch statement is for.

The following two examples are two different ways to write the same thing, one using a series of if statements, and the other using the switch statement:

```
if ($i == 0) {
    print "i equals 0";
}
if ($i == 1) {
    print "i equals 1";
}
if ($i == 2) {
    print "i equals 2";
}

switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
}
```

It is important to understand how the switch statement is executed in order to avoid mistakes. The switch statement executes line by line (actually, statement by statement). In the beginning, no code is executed. Only when a case statement is found with a value that matches the value of the switch expression does PHP begin to execute the statements. PHP continues to execute the statements until the end of the switch block, or the first time it sees a break statement. If you don't write a break statement at the end of a case's statement list, PHP will go on executing the statements of the following case. For example:

```
switch ($i) {
    case 0:
        print "i equals 0";
    case 1:
        print "i equals 1";
    case 2:
        print "i equals 2";
}
```

Here, if $i is equal to 0, PHP would execute all of the print statements! If $i is equal to 1, PHP would execute the last two print statements. You would get the expected behavior ('i equals 2' would be displayed) only if $i is equal to 2. Thus, it is important not to forget break statements (even though you may want to avoid supplying them on purpose under certain circumstances).

In a switch statement, the condition is evaluated only once and the result is compared to each case statement. In an elseif statement, the condition is evaluated again. If your condition is more complicated than a simple compare and/or is in a tight loop,

In a switch statement, the condition is evaluated only once and the result is compared to each case statement. In an elseif statement, the condition is evaluated again. If your condition is more complicated than a simple compare and/or is in a tight loop, a switch may be faster.

The statement list for a case can also be empty, which simply passes control into the statement list for the next case.

```
switch ($i) {
    case 0:
    case 1:
    case 2:
        print "i is less than 3 but not negative";
        break;
    case 3:
        print "i is 3";
}
```

A special case is the default case. This case matches anything that wasn't matched by the other cases, and should be the last case statement. For example:

```
switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
    default:
        print "i is not equal to 0, 1 or 2";
}
```

The case expression may be any expression that evaluates to a simple type, that is, integer or floating-point numbers and strings. Arrays or objects cannot be used here unless they are dereferenced to a simple type.

The alternative syntax for control structures is supported with switches. For more information, see Alternative syntax for control structures .

```
switch ($i):
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
    default:
        print "i is not equal to 0, 1 or 2";
endswitch;
```

---

declare

The declare construct is used to set execution directives for a block of code. The syntax of declare is similar to the syntax of other flow control constructs:

```
declare (directive) statement
```

The directive section allows the behavior of the declare block to be set. Currently only one directive is recognized: the ticks directive. (See below for more information on the ticks directive)

The statement part of the declare block will be executed -- how it is executed and what side effects occur during execution may depend on the directive set in the directive block.

Ticks

A tick is an event that occurs for every N low-level statements executed by the parser within the declare block. The value for N is specified using ticks=N within the declare blocks's directive section.

The event(s) that occur on each tick are specified using the **register_tick_function()**. See the example below for more details. Note that more than one event can occur for each tick.

Example 12-1. Profile a section of PHP code

```
<?php
// A function that records the time when it is called
function profile ($dump = FALSE)
{
    static $profile;

    // Return the times stored in profile, then erase it
```

```
    // Return the times stored in profile, then erase it
    if ($dump) {
        $temp = $profile;
        unset ($profile);
        return ($temp);
    }

    $profile[] = microtime ();
}

// Set up a tick handler
register_tick_function("profile");

// Initialize the function before the declare block
profile ();

// Run a block of code, throw a tick every 2nd statement
declare (ticks=2) {
    for ($x = 1; $x < 50; ++$x) {
        echo similar_text (md5($x), md5($x*$x)), "<br />;";
    }
}

// Display the data stored in the profiler
print_r (profile (TRUE));
?>
```

The example profiles the PHP code within the 'declare' block, recording the time at which every second low-level statement in the block was executed. This information can then be used to find the slow areas within particular segments of code. This process can be performed using other methods: using ticks is more convenient and easier to implement.

Ticks are well suited for debugging, implementing simple multitasking, backgrounded I/O and many other tasks.

See also **register_tick_function()** and **unregister_tick_function()**.

---

return

If called from within a function, the **return()** statement immediately ends execution of the current function, and returns its argument as the value of the function call. **return()** will also end the execution of an **eval()** statement or script file.

If called from the global scope, then execution of the current script file is ended. If the current script file was **include()**ed or **require()**ed, then control is passed back to the calling file. Furthermore, if the current script file was **include()**ed, then the value given to **return()** will be returned as the value of the **include()** call. If **return()** is called from within the main script file, then script execution ends. If the current script file was named by the auto_prepend_file or auto_append_file configuration options in the configuration file, then that script file's execution is ended.

For more information, see Returning values.

> **Note:** Note that since **return()** is a language construct and not a function, the parentheses surrounding its arguments are not required--in fact, it is more common to leave them out than to use them, although it doesn't matter one way or the other.

---

require()

The **require()** statement includes and evaluates the specific file.

require() includes and evaluates a specific file. Detailed information on how this inclusion works is described in the documentation for **include()**.

require() and **include()** are identical in every way except how they handle failure. **include()** produces a Warning while **require()** results in a Fatal Error. In other words, don't hesitate to use **require()** if you want a missing file to halt processing of the page. **include()** does not behave this way, the script will continue regardless. Be sure to have an appropriate include_path setting as well.

Example 12-2. Basic require() examples

```
<?php

require 'prepend.php';

require $somefile;

require ('somefile.txt');

?>
```

See the **include()** documentation for more examples.

> **Note:** Prior to PHP 4.0.2, the following applies: **require()** will always attempt to read the target file, even if the line it's on never executes. The conditional statement won't affect **require()**. However, if the line on which the **require()** occurs is not executed, neither will any of the code in the target file be executed. Similarly, looping structures do not affect the behaviour of **require()**. Although the code contained in the target file is still subject to the loop, the **require()** itself happens only once.

Warning
Windows versions of PHP prior to PHP 4.3 do not support accessing remote files via this function, even if allow_url_fopen is enabled.

See also **include()**, **require_once()**, **include_once()**, **eval()**, **file()**, **readfile()**, **virtual()** and include_path.

---

include()

include()

The **include()** statement includes and evaluates the specified file.

The documentation below also applies to **require()**. The two constructs are identical in every way except how they handle failure. **include()** produces a Warning while **require()** results in a Fatal Error. In other words, use **require()** if you want a missing file to halt processing of the page. **include()** does not behave this way, the script will continue regardless. Be sure to have an appropriate include_path setting as well.

When a file is included, the code it contains inherits the variable scope of the line on which the include occurs. Any variables available at that line in the calling file will be available within the called file, from that point forward.

Example 12-3. Basic include() example

```
vars.php
<?php

$color = 'green';
$fruit = 'apple';

?>

test.php
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

If the include occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function. So, it will follow the variable scope of that function.

Example 12-4. Including within functions

```
<?php

function foo()
{
global $color;

    include 'vars.php';

    echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so     *
 * $fruit is NOT available outside of this  *
 * scope.  $color is because we declared it *
 * as global.                               */

foo();                    // A green apple
echo "A $color $fruit";   // A green

?>
```

When a file is included, parsing drops out of PHP mode and into HTML mode at the beginning of the target file, and resumes again at the end. For this reason, any code inside the target file which should be executed as PHP code must be enclosed within valid PHP start and end tags.

If "URL fopen wrappers" are enabled in PHP (which they are in the default configuration), you can specify the file to be included using an URL (via HTTP or other supported wrapper - see Appendix I for a list of protocols) instead of a local pathname. If the target server interprets the target file as PHP code, variables may be passed to the included file using an URL request string as used with HTTP GET. This is not strictly speaking the same thing as including the file and having it inherit the parent file's variable scope; the script is actually being run on the remote server and the result is then being included into the local script.

Warning
Windows versions of PHP prior to PHP 4.3 do not support accessing remote files via this function, even if allow_url_fopen is enabled.

Example 12-5. include() through HTTP

```
<?php

/* This example assumes that www.example.com is configured to parse .php *
 * files and not .txt files. Also, 'Works' here means that the variables *
 * $foo and $bar are available within the included file.                  */

// Won't work; file.txt wasn't handled by www.example.com as PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// Won't work; looks for a file named 'file.php?foo=1&bar=2' on the
// local filesystem.
include 'file.php?foo=1&bar=2';

// Works.
```

```
// Works.
include 'http://www.example.com/file.php?foo=1&bar=2';

$foo = 1;
$bar = 2;
include 'file.txt';  // Works.
include 'file.php';  // Works.

?>
```

See also Remote files, **fopen()** and **file()** for related information.

Because **include()** and **require()** are special language constructs, you must enclose them within a statement block if it's inside a conditional block.

Example 12-6. include() and conditional blocks

```
<?php

// This is WRONG and will not work as desired.
if ($condition)
    include $file;
else
    include $other;


// This is CORRECT.
if ($condition) {
    include $file;
} else {
    include $other;
}

?>
```

Handling Returns: It is possible to execute a **return()** statement inside an included file in order to terminate processing in that file and return to the script which called it. Also, it's possible to return values from included files. You can take the value of the include call as you would a normal function.

> **Note:** In PHP 3, the return may not appear inside a block unless it's a function block, in which case the **return()** applies to that function and not the whole file.

Example 12-7. include() and the return() statement

```
return.php
<?php

$var = 'PHP';

return $var;

?>

noreturn.php
<?php

$var = 'PHP';

?>

testreturns.php
<?php

$foo = include 'return.php';

echo $foo; // prints 'PHP'

$bar = include 'noreturn.php';

echo $bar; // prints 1

?>
```

$bar is the value 1 because the include was successful. Notice the difference between the above examples. The first uses **return()** within the included file while the other does not. A few other ways to "include" files into variables are with **fopen()**, **file()** or by using **include()** along with Output Control Functions.

See also **require()**, **require_once()**, **include_once()**, **readfile()**, **virtual()**, and include_path.

---

require_once()

The **require_once()** statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the **require()** statement, with the only difference being that if the code from a file has already been included, it will not be included again. See the documentation for **require()** for more information on how this statement works.

require_once() should be used in cases where the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.

For examples on using **require_once()** and **include_once()**, look at the PEAR code included in the latest PHP source code distributions.

distributions.

Note: **require_once()** was added in PHP 4.0.1pl2

Note: Be aware, that the behaviour of **require_once()** and **include_once()** may not be what you expect on a non case sensitive operating system (such as Windows).

Example 12-8. require_once() is case sensitive

```
require_once("a.php"); // this will include a.php
require_once("A.php"); // this will include a.php again on Windows!
```

Warning
Windows versions of PHP prior to PHP 4.3 do not support accessing remote files via this function, even if allow_url_fopen is enabled.

See also: **require()**, **include()**, **include_once()**, **get_required_files()**, **get_included_files()**, **readfile()**, and **virtual()**.

---

include_once()

The **include_once()** statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the **include()** statement, with the only difference being that if the code from a file has already been included, it will not be included again. As the name suggests, it will be included just once.

include_once() should be used in cases where the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.

For more examples on using **require_once()** and **include_once()**, look at the PEAR code included in the latest PHP source code distributions.

Note: **include_once()** was added in PHP 4.0.1pl2

Note: Be aware, that the behaviour of **include_once()** and **require_once()** may not be what you expect on a non case sensitive operating system (such as Windows).

Example 12-9. include_once() is case sensitive

```
include_once("a.php"); // this will include a.php
include_once("A.php"); // this will include a.php again on Windows!
```

Warning
Windows versions of PHP prior to PHP 4.3 do not support accessing remote files via this function, even if allow_url_fopen is enabled.

See also **include()**, **require()**, **require_once()**, **get_required_files()**, **get_included_files()**, **readfile()**, and **virtual()**.

Chapter 13. Functions

User-defined functions

A function may be defined using syntax such as the following:

```
function foo ($arg_1, $arg_2, ..., $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
```

Any valid PHP code may appear inside a function, even other functions and class definitions.

In PHP 3, functions must be defined before they are referenced. No such requirement exists in PHP 4. Except when a function is conditionally defined such as shown in the two examples below.

When a function is defined in a conditional manner such as the two examples shown. Its definition must be processed prior to being called.

Example 13-1. Conditional functions

```
<?php

$makefoo = true;

/* We can't call foo() from here
   since it doesn't exist yet,
   but we can call bar() */

bar();

if ($makefoo) {
  function foo ()
  {
    echo "I don't exist until program execution reaches me.\n";
  }
}

/* Now we can safely call foo()
```

```
/* Now we can safely call foo()
   since $makefoo evaluated to true */

if ($makefoo) foo();

function bar() {
{
  echo "I exist immediately upon program start.\n";
}

?>
```

Example 13-2. Functions within functions

```
<?php
function foo()
{
  function bar()
  {
    echo "I don't exist until foo() is called.\n";
  }
}

/* We can't call bar() yet
   since it doesn't exist. */

foo();

/* Now we can call bar(),
   foo()'s processesing has
   made it accessable. */

bar();

?>
```

PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.

PHP 3 does not support variable numbers of arguments to functions, although default arguments are supported (see Default argument values for more information). PHP 4 supports both: see Variable-length argument lists and the function references for **func_num_args()**, **func_get_arg()**, and **func_get_args()** for more information.

Function arguments

Information may be passed to functions via the argument list, which is a comma-delimited list of variables and/or constants.

PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are supported only in PHP 4 and later; see Variable-length argument lists and the function references for **func_num_args()**, **func_get_arg()**, and **func_get_args()** for more information. A similar effect can be achieved in PHP 3 by passing an array of arguments to a function:

```
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

Making arguments be passed by reference

By default, function arguments are passed by value (so that if you change the value of the argument within the function, it does not get changed outside of the function). If you wish to allow a function to modify its arguments, you must pass them by reference.

If you want an argument to a function to always be passed by reference, you can prepend an ampersand (&) to the argument name in the function definition:

```
function add_some_extra(&$string)
{
    $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);
echo $str;    // outputs 'This is a string, and something extra.'
```

Default argument values

A function may define C++-style default values for scalar arguments as follows:

```
function makecoffee ($type = "cappuccino")
{
    return "Making a cup of $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

The output from the above snippet is:

The output from the above snippet is:

```
Making a cup of cappuccino.
Making a cup of espresso.
```

The default value must be a constant expression, not (for example) a variable or class member.

Note that when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected. Consider the following code snippet:

```
function makeyogurt ($type = "acidophilus", $flavour)
{
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");   // won't work as expected
```

The output of the above example is:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Making a bowl of raspberry .
```

Now, compare the above with this:

```
function makeyogurt ($flavour, $type = "acidophilus")
{
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");   // works as expected
```

The output of this example is:

```
Making a bowl of acidophilus raspberry.
```

Variable-length argument lists

PHP 4 has support for variable-length argument lists in user-defined functions. This is really quite easy, using the **func_num_args()**, **func_get_arg()**, and **func_get_args()** functions.

No special syntax is required, and argument lists may still be explicitly provided with function definitions and will behave as normal.

Returning values

Values are returned by using the optional return statement. Any type may be returned, including lists and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called. See **return()** for more information.

```
function square ($num)
{
    return $num * $num;
}
echo square (4);   // outputs '16'.
```

You can't return multiple values from a function, but similar results can be obtained by returning a list.

```
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
```

To return a reference from a function, you have to use the reference operator & in both the function declaration and when assigning the returned value to a variable:

```
function &returns_reference()
{
    return $someref;
}

$newref =& returns_reference();
```

For more information on references, please check out References Explained.

---

old_function

The old_function statement allows you to declare a function using a syntax identical to PHP/FI2 (except you must replace 'function' with 'old_function'.

The old_function statement allows you to declare a function using a syntax identical to PHP/FI2 (except you must replace 'function' with 'old_function'.

This is a deprecated feature, and should only be used by the PHP/FI2->PHP 3 convertor.

Warning
Functions declared as old_function cannot be called from PHP's internal code. Among other things, this means you can't use them in functions such as **usort()**, **array_walk()**, and **register_shutdown_function()**. You can get around this limitation by writing a wrapper function (in normal PHP 3 form) to call the old_function.

Variable functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

Variable functions won't work with language constructs such as **echo()**, **print()**, **unset()**, **isset()**, **empty()**, **include()**, **require()** and the like. You need to use your own wrapper function to utilize any of these constructs as variable functions.

Example 13-3. Variable function example

```
<?php
function foo()
{
    echo "In foo()<br>\n";
}

function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br>\n";
}

// This is a wrapper function around echo
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // This calls foo()

$func = 'bar';
$func('test');  // This calls bar()

$func = 'echoit';
$func('test');  // This calls echoit()
?>
```

You can also call an object's method by using the variable functions feature.

Example 13-4. Variable method example

```
<?php
class Foo
{
    function Var()
    {
        $name = 'Bar';
        $this->$name(); // This calls the Bar() method
    }

    function Bar()
    {
        echo "This is Bar";
    }
}

$foo = new Foo();
$funcname = "Var";
$foo->$varname();   // This calls $foo->Var()

?>
```

See also **call_user_func()**, variable variables and **function_exists()**.

Chapter 14. Classes and Objects

class

A class is a collection of variables and functions working with these variables. A class is defined using the following syntax:

```
<?php
class Cart
{
    var $items;  // Items in our shopping cart

    // Add $num articles of $artnr to the cart
```

```
    // Add $num articles of $artnr to the cart

    function add_item ($artnr, $num)
    {
        $this->items[$artnr] += $num;
    }

    // Take $num articles of $artnr out of the cart

    function remove_item ($artnr, $num)
    {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } else {
            return false;
        }
    }
}
?>
```

This defines a class named Cart that consists of an associative array of articles in the cart and two functions to add and remove items from this cart.

Caution
The following cautionary notes are valid for PHP 4. The name stdClass is used interally by Zend and is reserved. You cannot have a class named stdClass in PHP. The function names __sleep and __wakeup are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them. See below for more information. PHP reserves all function names starting with __ as magical. It is recommended that you do not use function names with __ in PHP unless you want some documented magic functionality.

Note: In PHP 4, only constant initializers for var variables are allowed. To initialize variables with non-constant values, you need an initialization function which is called automatically when an object is being constructed from the class. Such a function is called a constructor (see below).

```
<?php
/* None of these will work in PHP 4. */
class Cart
{
    var $todays_date = date("Y-m-d");
    var $name = $firstname;
    var $owner = 'Fred ' . 'Jones';
    var $items = array("VCR", "TV");
}

/* This is how it should be done. */
class Cart
{
    var $todays_date;
    var $name;
    var $owner;
    var $items;

    function Cart()
    {
        $this->todays_date = date("Y-m-d");
        $this->name = $GLOBALS['firstname'];
        /* etc. . . */
    }
}
?>
```

Classes are types, that is, they are blueprints for actual variables. You have to create a variable of the desired type with the new operator.

```
<?php
$cart = new Cart;
$cart->add_item("10", 1);

$another_cart = new Cart;
$another_cart->add_item("0815", 3);
```

This creates the objects $cart and $another_cart, both of the class Cart. The function add_item() of the $cart object is being called to add 1 item of article number 10 to the $cart. 3 items of article number 0815 are being added to $another_cart.

Both, $cart and $another_cart, have functions add_item(), remove_item() and a variable items. These are distinct functions and variables. You can think of the objects as something similar to directories in a filesystem. In a filesystem you can have two different files README.TXT, as long as they are in different directories. Just like with directories where you'll have to type the full pathname in order to reach each file from the toplevel directory, you have to specify the complete name of the function you want to call: In PHP terms, the toplevel directory would be the global namespace, and the pathname separator would be ->. Thus, the names $cart->items and $another_cart->items name two different variables. Note that the variable is named $cart->items, not $cart->$items, that is, a variable name in PHP has only a single dollar sign.

```
// correct, single $
```

```
// correct, single $
$cart->items = array("10" => 1);

// invalid, because $cart->$items becomes $cart->""
$cart->$items = array("10" => 1);

// correct, but may or may not be what was intended:
// $cart->$myvar becomes $cart->items
$myvar = 'items';
$cart->$myvar = array("10" => 1);
```

Within a class definition, you do not know under which name the object will be accessible in your program: at the time the Cart class was written, it was unknown that the object will be named $cart or $another_cart later. Thus, you cannot write $cart->items within the Cart class itself. Instead, in order to be able to access it's own functions and variables from within a class, one can use the pseudo-variable $this which can be read as 'my own' or 'current object'. Thus, '$this->items[$artnr] += $num' can be read as 'add $num to the $artnr counter of my own items array' or 'add $num to the $artnr counter of the items array within the current object'.

> **Note:** There are some nice functions to handle classes and objects. You might want to take a look at the Class/Object Functions

---

extends

Often you need classes with similar variables and functions to another existing class. In fact, it is good practice to define a generic class which can be used in all your projects and adapt this class for the needs of each of your specific projects. To facilitate this, classes can be extensions of other classes. The extended or derived class has all variables and functions of the base class (this is called 'inheritance' despite the fact that nobody died) and what you add in the extended definition. It is not possible to substract from a class, that is, to undefine any existing functions or variables. An extended class is always dependent on a single base class, that is, multiple inheritance is not supported. Classes are extended using the keyword 'extends'.

```
class Named_Cart extends Cart
{
    var $owner;

    function set_owner ($name)
    {
        $this->owner = $name;
    }
}
```

This defines a class Named_Cart that has all variables and functions of Cart plus an additional variable $owner and an additional function set_owner(). You create a named cart the usual way and can now set and get the carts owner. You can still use normal cart functions on named carts:

```
$ncart = new Named_Cart;    // Create a named cart
$ncart->set_owner("kris");  // Name that cart
print $ncart->owner;        // print the cart owners name
$ncart->add_item("10", 1);  // (inherited functionality from cart)
```

This is also called a "parent-child" relationship. You create a class, parent, and use extends to create a new class based on the parent class: the child class. You can even use this new child class and create another class based on this child class.

> **Note:** Classes must be defined before they are used! If you want the class Named_Cart to extend the class Cart, you will have to define the class Cart first. If you want to create another class called Yellow_named_cart based on the class Named_Cart you have to define Named_Cart first. To make it short: the order in which the classes are defined is important.

Constructors

Caution
In PHP 3 and PHP 4 constructors behave differently. The PHP 4 semantics are strongly preferred.

Constructors are functions in a class that are automatically called when you create a new instance of a class with new. In PHP 3, a function becomes a constructor when it has the same name as the class. In PHP 4, a function becomes a constructor, when it has the same name as the class it is defined in - the difference is subtle, but crucial (see below).

```
// Works in PHP 3 and PHP 4.
class Auto_Cart extends Cart
{
    function Auto_Cart()
    {
        $this->add_item ("10", 1);
    }
}
```

This defines a class Auto_Cart that is a Cart plus a constructor which initializes the cart with one item of article number "10" each time a new Auto_Cart is being made with "new". Constructors can take arguments and these arguments can be optional, which makes them much more useful. To be able to still use the class without parameters, all parameters to constructors should be made optional by providing default values.

```
// Works in PHP 3 and PHP 4.
class Constructor_Cart extends Cart
{
    function Constructor_Cart($item = "10", $num = 1)
    {
        $this->add_item ($item, $num);
    }
}

// Shop the same old boring stuff.

$default_cart = new Constructor_Cart;

// Shop for real...

$different_cart = new Constructor_Cart("20", 17);
```

You also can use the @ operator to mute errors occuring in the constructor, e.g. @new.

Caution
In PHP 3, derived classes and constructors have a number of limitations. The following examples should be read carefully to understand these limitations.

```
class A
{
    function A()
    {
      echo "I am the constructor of A.<br>\n";
    }
}

class B extends A
{
    function C()
    {
        echo "I am a regular function.<br>\n";
    }
}

// no constructor is being called in PHP 3.
$b = new B;
```

In PHP 3, no constructor is being called in the above example. The rule in PHP 3 is: 'A constructor is a function of the same name as the class.'. The name of the class is B, and there is no function called B() in class B. Nothing happens.

This is fixed in PHP 4 by introducing another rule: If a class has no constructor, the constructor of the base class is being called, if it exists. The above example would have printed 'I am the constructor of A.
' in PHP 4.

```
class A
{
    function A()
    {
        echo "I am the constructor of A.<br>\n";
    }

    function B()
    {
        echo "I am a regular function named B in class A.<br>\n";
        echo "I am not a constructor in A.<br>\n";
    }
}

class B extends A
{
    function C()
    {
        echo "I am a regular function.<br>\n";
    }
}

// This will call B() as a constructor.
$b = new B;
```

In PHP 3, the function B() in class A will suddenly become a constructor in class B, although it was never intended to be. The rule in PHP 3 is: 'A constructor is a function of the same name as the class.'. PHP 3 does not care if the function is being defined in class B, or if it has been inherited.

This is fixed in PHP 4 by modifying the rule to: 'A constructor is a function of the same name as the class it is being defined in.'. Thus in PHP 4, the class B would have no constructor function of its own and the constructor of the base class would have been called, printing 'I am the constructor of A.
'.

Caution
Neither PHP 3 nor PHP 4 call constructors of the base class automatically from a constructor of a derived class. It is your responsibility to propagate the call to constructors upstream where appropriate.

Note: There are no destructors in PHP 3 or PHP 4. You may use **register_shutdown_function()** instead to simulate

responsibility to propagate the call to constructors upstream where appropriate.

Note: There are no destructors in PHP 3 or PHP 4. You may use **register_shutdown_function()** instead to simulate most effects of destructors.

Destructors are functions that are called automatically when an object is destroyed, either with **unset()** or by simply going out of scope. There are no destructors in PHP.

::

Caution
The following is valid for PHP 4 only.

Sometimes it is useful to refer to functions and variables in base classes or to refer to functions in classes that have not yet any instances. The :: operator is being used for this.

```
class A
{
    function example()
    {
        echo "I am the original function A::example().<br>\n";
    }
}

class B extends A
{
    function example()
    {
        echo "I am the redefined function B::example().<br>\n";
        A::example();
    }
}

// there is no object of class A.
// this will print
//   I am the original function A::example().<br>
A::example();

// create an object of class B.
$b = new B;

// this will print
//   I am the redefined function B::example().<br>
//   I am the original function A::example().<br>
$b->example();
```

The above example calls the function example() in class A, but there is no object of class A, so that we cannot write $a->example() or similar. Instead we call example() as a 'class function', that is, as a function of the class itself, not any object of that class.

There are class functions, but there are no class variables. In fact, there is no object at all at the time of the call. Thus, a class function may not use any object variables (but it can use local and global variables), and it may no use $this at all.

In the above example, class B redefines the function example(). The original definition in class A is shadowed and no longer available, unless you are refering specifically to the implementation of example() in class A using the ::-operator. Write A::example() to do this (in fact, you should be writing parent::example(), as shown in the next section).

In this context, there is a current object and it may have object variables. Thus, when used from WITHIN an object function, you may use $this and object variables.

---

parent

You may find yourself writing code that refers to variables and functions in base classes. This is particularly true if your derived class is a refinement or specialisation of code in your base class.

Instead of using the literal name of the base class in your code, you should be using the special name parent, which refers to the name of your base class as given in the extends declaration of your class. By doing this, you avoid using the name of your base class in more than one place. Should your inheritance tree change during implementation, the change is easily made by simply changing the extends declaration of your class.

```
class A
{
    function example()
    {
        echo "I am A::example() and provide basic functionality.<br>\n";
    }
}

class B extends A
{
    function example()
    {
        echo "I am B::example() and provide additional functionality.<br>\n";
        parent::example();
    }
}

$b = new B;

// This will call B::example(), which will in turn call A::example().
$b->example();
```

```
// This will call B::example(), which will in turn call A::example().
$b->example();
```

Serializing objects - objects in sessions

> **Note:** In PHP 3, objects will lose their class association throughout the process of serialization and unserialization. The resulting variable is of type object, but has no class and no methods, thus it is pretty useless (it has become just like an array with a funny syntax).

Caution
The following information is valid for PHP 4 only.

serialize() returns a string containing a byte-stream representation of any value that can be stored in PHP. **unserialize()** can use this string to recreate the original variable values. Using serialize to save an object will save all variables in an object. The functions in an object will not be saved, only the name of the class.

In order to be able to **unserialize()** an object, the class of that object needs to be defined. That is, if you have an object $a of class A on page1.php and serialize this, you'll get a string that refers to class A and contains all values of variabled contained in $a. If you want to be able to unserialize this on page2.php, recreating $a of class A, the definition of class A must be present in page2.php. This can be done for example by storing the class defintion of class A in an include file and including this file in both page1.php and page2.php.

```
classa.inc:
  class A
  {
      var $one = 1;

      function show_one()
      {
          echo $this->one;
      }
  }

page1.php:
  include("classa.inc");

  $a = new A;
  $s = serialize($a);
  // store $s somewhere where page2.php can find it.
  $fp = fopen("store", "w");
  fputs($fp, $s);
  fclose($fp);

page2.php:
  // this is needed for the unserialize to work properly.
  include("classa.inc");

  $s = implode("", @file("store"));
  $a = unserialize($s);

  // now use the function show_one() of the $a object.
  $a->show_one();
```

If you are using sessions and use **session_register()** to register objects, these objects are serialized automatically at the end of each PHP page, and are unserialized automatically on each of the following pages. This basically means that these objects can show up on any of your pages once they become part of your session.

It is strongly recommended that you include the class definitions of all such registered objects on all of your pages, even if you do not actually use these classes on all of your pages. If you don't and an object is being unserialized without its class definition being present, it will lose its class association and become an object of class stdClass without any functions available at all, that is, it will become quite useless.

So if in the example above $a became part of a session by running session_register("a"), you should include the file classa.inc on all of your pages, not only page1.php and page2.php.

The magic functions __sleep and __wakeup

serialize() checks if your class has a function with the magic name __sleep. If so, that function is being run prior to any serialization. It can clean up the object and is supposed to return an array with the names of all variables of that object that should be serialized.

The intended use of __sleep is to close any database connections that object may have, committing pending data or perform similar cleanup tasks. Also, the function is useful if you have very large objects which need not be saved completely.

Conversely, **unserialize()** checks for the presence of a function with the magic name __wakeup. If present, this function can reconstruct any resources that object may have.

The intended use of __wakeup is to reestablish any database connections that may have been lost during serialization and perform other reinitialization tasks.

References inside the constructor

Creating references within the constructor can lead to confusing results. This tutorial-like section helps you to avoid problems.

```
class Foo
{
    function Foo($name)
    {
```

```
{
    function Foo($name)
    {
        // create a reference inside the global array $globalref
        global $globalref;
        $globalref[] = &$this;
        // set name to passed value
        $this->setName($name);
        // and put it out
        $this->echoName();
    }

    function echoName()
    {
        echo "<br>",$this->name;
    }

    function setName($name)
    {
        $this->name = $name;
    }
}
```

Let us check out if there is a difference between $bar1 which has been created using the copy = operator and $bar2 which has been created using the reference =& operator...

```
$bar1 = new Foo('set in constructor');
$bar1->echoName();
$globalref[0]->echoName();

/* output:
set in constructor
set in constructor
set in constructor */

$bar2 =& new Foo('set in constructor');
$bar2->echoName();
$globalref[1]->echoName();

/* output:
set in constructor
set in constructor
set in constructor */
```

Apparently there is no difference, but in fact there is a very significant one: $bar1 and $globalref[0] are _NOT_ referenced, they are NOT the same variable. This is because "new" does not return a reference by default, instead it returns a copy.

> **Note:** There is no performance loss (since PHP 4 and up use reference counting) returning copies instead of references. On the contrary it is most often better to simply work with copies instead of references, because creating references takes some time where creating copies virtually takes no time (unless none of them is a large array or object and one of them gets changed and the other(s) one(s) subsequently, then it would be wise to use references to change them all concurrently).

To prove what is written above let us watch the code below.

```
// now we will change the name. what do you expect?
// you could expect that both $bar1 and $globalref[0] change their names...
$bar1->setName('set from outside');

// as mentioned before this is not the case.
$bar1->echoName();
$globalref[0]->echoName();

/* output:
set from outside
set in constructor */

// let us see what is different with $bar2 and $globalref[1]
$bar2->setName('set from outside');

// luckily they are not only equal, they are the same variable
// thus $bar2->name and $globalref[1]->name are the same too
$bar2->echoName();
$globalref[1]->echoName();

/* output:
set from outside
set from outside */
```

Another final example, try to understand it.

```
class A
{
    function A($i)
    {
        $this->value = $i;
        // try to figure out why we do not need a reference here
        $this->b = new B($this);
    }

    function createRef()
```

```
    }

    function createRef()
    {
        $this->c = new B($this);
    }

    function echoValue()
    {
        echo "<br>","class ",get_class($this),': ',$this->value;
    }
}


class B
{
    function B(&$a)
    {
        $this->a = &$a;
    }

    function echoValue()
    {
        echo "<br>","class ",get_class($this),': ',$this->a->value;
    }
}

// try to undestand why using a simple copy here would yield
// in an undesired result in the *-marked line
$a =& new A(10);
$a->createRef();

$a->echoValue();
$a->b->echoValue();
$a->c->echoValue();

$a->value = 11;

$a->echoValue();
$a->b->echoValue(); // *
$a->c->echoValue();

/*
output:
class A: 10
class B: 10
class B: 10
class A: 11
class B: 11
class B: 11
*/
```

Chapter 15. References Explained

What References Are

References in PHP are a means to access the same variable content by different names. They are not like C pointers, they are symbol table aliases. Note that in PHP, variable name and variable content are different, so the same content can have different names. The most close analogy is with Unix filenames and files - variable names are directory entries, while variable contents is the file itself. References can be thought of as hardlinking in Unix filesystem.

What References Do

PHP references allow you to make two variables to refer to the same content. Meaning, when you do:

```
$a =& $b
```

it means that $a and $b point to the same variable.

> **Note:** $a and $b are completely equal here, that's not $a is pointing to $b or vice versa, that's $a and $b pointing to the same place.

The same syntax can be used with functions, that return references, and with new operator (in PHP 4.0.4 and later):

```
$bar =& new fooclass();
$foo =& find_var ($bar);
```

> **Note:** Not using the & operator causes a copy of the object to be made. If you use $this in the class it will operate on the current instance of the class. The assignment without & will copy the instance (i.e. the object) and $this will operate on the copy, which is not always what is desired. Usually you want to have a single instance to work with, due to performance and memory consumption issues.
>
> While you can use the @ operator to mute any errors in the constructor when using it as @new, this does not work when using the &new statement. This is a limitation of the Zend Engine and will therefore result in a parser error.

The second thing references do is to pass variables by-reference. This is done by making a local variable in a function and a variable in the calling scope reference to the same content. Example:

```
function foo (&$var)
{
    $var++;
}

$a=5;
foo ($a);
```

will make $a to be 6. This happens because in the function foo the variable $var refers to the same content as $a. See also more detailed explanations about passing by reference.

The third thing reference can do is return by reference.

What References Are Not

As said before, references aren't pointers. That means, the following construct won't do what you expect:

```
function foo (&$var)
{
    $var =& $GLOBALS["baz"];
}
foo($bar);
```

What happens is that $var in foo will be bound with $bar in caller, but then it will be re-bound with $GLOBALS["baz"]. There's no way to bind $bar in the calling scope to something else using the reference mechanism, since $bar is not available in the function foo (it is represented by $var, but $var has only variable contents and not name-to-value binding in the calling symbol table).

Passing by Reference

You can pass variable to function by reference, so that function could modify its arguments. The syntax is as follows:

```
function foo (&$var)
{
    $var++;
}

$a=5;
foo ($a);
// $a is 6 here
```

Note that there's no reference sign on function call - only on function definition. Function definition alone is enough to correctly pass the argument by reference.

Following things can be passed by reference:

- Variable, i.e. foo($a)
- New statement, i.e. foo(new foobar())
- Reference, returned from a function, i.e.:

```
function &bar()
{
    $a = 5;
    return $a;
}
foo(bar());
```

See also explanations about returning by reference.

Any other expression should not be passed by reference, as the result is undefined. For example, the following examples of passing by reference are invalid:

```
function bar() // Note the missing &
{
    $a = 5;
    return $a;
}
foo(bar());

foo($a = 5) // Expression, not variable
foo(5) // Constant, not variable
```

These requirements are for PHP 4.0.4 and later.

Returning References

Returning by-reference is useful when you want to use a function to find which variable a reference should be bound to. When returning references, use this syntax:

```
function &find_var ($param)
{
    ...code...
    return $found_var;
```

```
{
    ...code...
    return $found_var;
}

$foo =& find_var ($bar);
$foo->x = 2;
```

In this example, the property of the object returned by the find_var function would be set, not the copy, as it would be without using reference syntax.

> **Note:** Unlike parameter passing, here you have to use & in both places - to indicate that you return by-reference, not a copy as usual, and to indicate that reference binding, rather than usual assignment, should be done for $foo.

Unsetting References

When you unset the reference, you just break the binding between variable name and variable content. This does not mean that variable content will be destroyed. For example:

```
$a = 1;
$b =& $a;
unset ($a);
```

won't unset $b, just $a.

Again, it might be useful to think about this as analogous to Unix **unlink** call.

Spotting References

Many syntax constructs in PHP are implemented via referencing mechanisms, so everything told above about reference binding also apply to these constructs. Some constructs, like passing and returning by-reference, are mentioned above. Other constructs that use references are:

---

global References

When you declare variable as **global $var** you are in fact creating reference to a global variable. That means, this is the same as:

```
$var =& $GLOBALS["var"];
```

That means, for example, that unsetting $var won't unset global variable.

$this

In an object method, $this is always reference to the caller object.

III. Features

Table of Contents

Chapter 16. HTTP authentication with PHP

The HTTP Authentication hooks in PHP are only available when it is running as an Apache module and is hence not available in the CGI version. In an Apache module PHP script, it is possible to use the **header()** function to send an "Authentication Required" message to the client browser causing it to pop up a Username/Password input window. Once the user has filled in a username and a password, the URL containing the PHP script will be called again with the predefined variables PHP_AUTH_USER, PHP_AUTH_PW, and PHP_AUTH_TYPE set to the user name, password and authentication type respectively. These predefined variables are found in the $_SERVER and $HTTP_SERVER_VARS arrays. Only "Basic" authentication is supported. See the **header()** function for more information.

> **PHP Version Note:** Autoglobals, such as $_SERVER, became available in PHP version 4.1.0. $HTTP_SERVER_VARS has been available since PHP 3.

An example script fragment which would force client authentication on a page is as follows:

Example 16-1. HTTP Authentication example

```
<?php
  if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
  } else {
    echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
    echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
  }
?>
```

> **Compatibility Note:** Please be careful when coding the HTTP header lines. In order to guarantee maximum

?>

> **Compatibility Note:** Please be careful when coding the HTTP header lines. In order to guarantee maximum compatibility with all clients, the keyword "Basic" should be written with an uppercase "B", the realm string must be enclosed in double (not single) quotes, and exactly one space should precede the 401 code in the HTTP/1.0 401 header line.

Instead of simply printing out PHP_AUTH_USER and PHP_AUTH_PW, as done in the above example, you may want to check the username and password for validity. Perhaps by sending a query to a database, or by looking up the user in a dbm file.

Watch out for buggy Internet Explorer browsers out there. They seem very picky about the order of the headers. Sending the WWW-Authenticate header before the HTTP/1.0 401 header seems to do the trick for now.

As of PHP 4.3.0, in order to prevent someone from writing a script which reveals the password for a page that was authenticated through a traditional external mechanism, the PHP_AUTH variables will not be set if external authentication is enabled for that particular page and safe mode is enabled. Regardless, REMOTE_USER can be used to identify the externally-authenticated user. So, you can use $_SERVER['REMOTE_USER'].

> **Configuration Note:** PHP uses the presence of an AuthType directive to determine whether external authentication is in effect.

Note, however, that the above does not prevent someone who controls a non-authenticated URL from stealing passwords from authenticated URLs on the same server.

Both Netscape Navigator and Internet Explorer will clear the local browser window's authentication cache for the realm upon receiving a server response of 401. This can effectively "log out" a user, forcing them to re-enter their username and password. Some people use this to "time out" logins, or provide a "log-out" button.

Example 16-2. HTTP Authentication example forcing a new name/password

```
<?php
  function authenticate() {
    header('WWW-Authenticate: Basic realm="Test Authentication System"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You must enter a valid login ID and password to access this resource\n";
    exit;
  }

  if (!isset($_SERVER['PHP_AUTH_USER']) ||
      ($_POST['SeenBefore'] == 1 && $_POST['OldAuth'] == $_SERVER['PHP_AUTH_USER'])) {
   authenticate();
  }
  else {
   echo "<p>Welcome: {$_SERVER['PHP_AUTH_USER']}<br>";
   echo "Old: {$_REQUEST['OldAuth']}";
   echo "<form action='{$_SERVER['PHP_SELF']}' METHOD='POST'>\n";
   echo "<input type='hidden' name='SeenBefore' value='1'>\n";
   echo "<input type='hidden' name='OldAuth' value='{$_SERVER['PHP_AUTH_USER']}'>\n";
   echo "<input type='submit' value='Re Authenticate'>\n";
   echo "</form></p>\n";
  }
?>
```

This behavior is not required by the HTTP Basic authentication standard, so you should never depend on this. Testing with Lynx has shown that Lynx does not clear the authentication credentials with a 401 server response, so pressing back and then forward again will open the resource as long as the credential requirements haven't changed. The user can press the '_' key to clear their authentication information, however.

Also note that this does not work using Microsoft's IIS server and the CGI version of PHP due to a limitation of IIS.

> **Note:** If safe mode is enabled, the uid of the script is added to the realm part of the WWW-Authenticate header.

Chapter 17. Cookies

PHP transparently supports HTTP cookies. Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the **setcookie()** function. Cookies are part of the HTTP header, so **setcookie()** must be called before any output is sent to the browser. This is the same limitation that **header()** has. You can use the output buffering functions to delay the script output until you have decided whether or not to set any cookies or send any headers.

Any cookies sent to you from the client will automatically be turned into a PHP variable just like GET and POST method data, depending on the register_globals and variables_order configuration variables. If you wish to assign multiple values to a single cookie, just add [] to the cookie name.

In PHP 4.1.0 and later, the $_COOKIE auto-global array will always be set with any cookies sent from the client. $HTTP_COOKIE_VARS is also set in earlier versions of PHP when the track_vars configuration variable is set.

For more details, including notes on browser bugs, see the **setcookie()** function.

Chapter 18. Handling file uploads

POST method uploads

PHP is capable of receiving file uploads from any RFC-1867 compliant browser (which includes Netscape Navigator 3 or later, Microsoft Internet Explorer 3 with a patch from Microsoft, or later without a patch). This feature lets people upload both text and binary files. With PHP's authentication and file manipulation functions, you have full control over who is allowed to upload and what is to be done with the file once it has been uploaded.

> **Related Configurations Note:** See also the file_uploads, upload_max_filesize, upload_tmp_dir, and post_max_size directives in php.ini

Note that PHP also supports PUT-method file uploads as used by Netscape Composer and W3C's Amaya clients. See the PUT Method Support for more details.

A file upload screen can be built by creating a special form which looks something like this:

Method Support for more details.

A file upload screen can be built by creating a special form which looks something like this:

Example 18-1. File Upload Form

```
<form enctype="multipart/form-data" action="_URL_" method="post">
<input type="hidden" name="MAX_FILE_SIZE" value="30000">
Send this file: <input name="userfile" type="file">
<input type="submit" value="Send File">
</form>
```

The _URL_ should point to a PHP file. The MAX_FILE_SIZE hidden field must precede the file input field and its value is the maximum filesize accepted. The value is in bytes.

Warning
The MAX_FILE_SIZE is advisory to the browser. It is easy to circumvent this maximum. So don't count on it that the browser obeys your wish! The PHP-settings for maximum-size, however, cannot be fooled.

The Variables defined for uploaded files differs depending on the PHP version and configuration. The autoglobal $_FILES exists as of PHP 4.1.0 The $HTTP_POST_FILES array has existed since PHP 4.0.0. These arrays will contain all your uploaded file information. Using $_FILES is preferred. If the PHP directive register_globals is on, related variable names will also exist. register_globals defaults to off as of PHP 4.2.0.

The contents of $_FILES from our example script is as follows. Note that this assumes the use of the file upload name userfile, as used in the example script above.

$_FILES['userfile']['name']

The original name of the file on the client machine.

$_FILES['userfile']['type']

The mime type of the file, if the browser provided this information. An example would be "image/gif".

$_FILES['userfile']['size']

The size, in bytes, of the uploaded file.

$_FILES['userfile']['tmp_name']

The temporary filename of the file in which the uploaded file was stored on the server.

$_FILES['userfile']['error']

The error code associated with this file upload. ['error'] was added in PHP 4.2.0

Note: In PHP versions prior 4.1.0 this was named $HTTP_POST_FILES and it's not an autoglobal variable like $_FILES is. PHP 3 does not support $HTTP_POST_FILES.

When register_globals is turned on in php.ini, additional variables are available. For example, $userfile_name will equal $_FILES['userfile']['name'], $userfile_type will equal $_FILES['userfile']['type'], etc. Keep in mind that as of PHP 4.2.0, register_globals defaults to off. It's preferred to not rely on this directive.

Files will by default be stored in the server's default temporary directory, unless another location has been given with the upload_tmp_dir directive in php.ini. The server's default directory can be changed by setting the environment variable TMPDIR in the environment in which PHP runs. Setting it using **putenv()** from within a PHP script will not work. This environment variable can also be used to make sure that other operations are working on uploaded files, as well.

Example 18-2. Validating file uploads

See also the function entries for **is_uploaded_file()** and **move_uploaded_file()** for further information. The following example will process the file upload that came from a form.

```
<?php
// In PHP earlier then 4.1.0, $HTTP_POST_FILES should be used instead of $_FILES.
// In PHP earlier then 4.0.3, use copy() and is_uploaded_file() instead of move_uploaded_file

$uploaddir = '/var/www/uploads/';

print "<pre>";
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploaddir . $_FILES['userfile']['name'])) {
    print "File is valid, and was successfully uploaded.  Here's some more debugging info:\n";
    print_r($_FILES);
} else {
    print "Possible file upload attack!  Here's some debugging info:\n";
    print_r($_FILES);
}

?>
```

The PHP script which receives the uploaded file should implement whatever logic is necessary for determining what should be done with the uploaded file. You can for example use the $_FILES['userfile']['size'] variable to throw away any files that are either too small or too big. You could use the $_FILES['userfile']['type'] variable to throw away any files that didn't match a certain type criteria. As of PHP 4.2.0, you could use $_FILES['userfile']['error'] and plan your logic according to the error codes. Whatever the logic, you should either delete the file from the temporary directory or move it elsewhere.

The file will be deleted from the temporary directory at the end of the request if it has not been moved away or renamed.

Error Messages Explained

Since PHP 4.2.0, PHP returns an appropriate error code along with the file array. The error code can be found in the ['error'] segment of the file array that is created during the file upload by PHP. In otherwords, the error might be found in $_FILES['userfile']['error'].

UPLOAD_ERR_OK

Value: 0; There is no error, the file uploaded with success.

UPLOAD_ERR_INI_SIZE

Value: 0; There is no error, the file uploaded with success.

UPLOAD_ERR_INI_SIZE

Value: 1; The uploaded file exceeds the upload_max_filesize directive in php.ini.

UPLOAD_ERR_FORM_SIZE

Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was specified in the html form.

UPLOAD_ERR_PARTIAL

Value: 3; The uploaded file was only partially uploaded.

UPLOAD_ERR_NO_FILE

Value: 4; No file was uploaded.

Note: These became PHP constants in PHP 4.3.0

Common Pitfalls

The MAX_FILE_SIZE item cannot specify a file size greater than the file size that has been set in the upload_max_filesize ini-setting. The default is 2 Megabytes.

If memory limit is enabled, larger memory_limit may be needed. Make sure to set memory_limit large enough.

If max_execution_time is set too small, script execution may be exceeded the value. Make sure to set max_execution_time large enough.

If post_max_size set too small, large files cannot be uploaded. Make sure to set post_max_size large enough.

Not validating which file you operate on may mean that users can access sensitive information in other directories.

Please note that the CERN httpd seems to strip off everything starting at the first whitespace in the content-type mime header it gets from the client. As long as this is the case, CERN httpd will not support the file upload feature.

Due to the large amount of directory listing styles we cannot guarantee that files with exotic names (like containing spaces) are handled properly.

Uploading multiple files

Multiple files can be uploaded using different name for input.

It is also possible to upload multiple files simultaneously and have the information organized automatically in arrays for you. To do so, you need to use the same array submission syntax in the HTML form as you do with multiple selects and checkboxes:

Note: Support for multiple file uploads was added in version 3.0.10.

Example 18-3. Uploading multiple files

```
<form action="file-upload.php" method="post" enctype="multipart/form-data">
  Send these files:<br>
  <input name="userfile[]" type="file"><br>
  <input name="userfile[]" type="file"><br>
  <input type="submit" value="Send files">
</form>
```

When the above form is submitted, the arrays $_FILES['userfile'], $_FILES['userfile']['name'], and $_FILES['userfile']['size'] will be initialized (as well as in $HTTP_POST_FILES for PHP version prior 4.1.0). When register_globals is on, globals for uploaded files are also initialized. Each of these will be a numerically indexed array of the appropriate values for the submitted files.

For instance, assume that the filenames /home/test/review.html and /home/test/xwp.out are submitted. In this case, $_FILES['userfile']['name'][0] would contain the value review.html, and $_FILES['userfile']['name'][1] would contain the value xwp.out. Similarly, $_FILES['userfile']['size'][0] would contain review.html's filesize, and so forth.

$_FILES['userfile']['name'][0], $_FILES['userfile']['tmp_name'][0], $_FILES['userfile']['size'][0], and $_FILES['userfile']['type'][0] are also set.

PUT method support

PUT method support has changed between PHP 3 and PHP 4. In PHP 4, one should use the standard input stream to read the contents of an HTTP PUT.

Example 18-4. Saving HTTP PUT files with PHP 4

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://stdin","r");

/* Open a file for writting */
$fp = fopen("myputfile.ext","w");

/* Read the data 1kb at a time
   and write to the file */
while ($data = fread($putdata,1024))
  fwrite($fp,$data);

/* Close the streams */
fclose($fp);
fclose($putdata);
?>
```

Note: All documentation below applies to PHP 3 only.

PHP provides support for the HTTP PUT method used by clients such as Netscape Composer and W3C Amaya. PUT requests are much simpler than a file upload and they look something like this:

PHP provides support for the HTTP PUT method used by clients such as Netscape Composer and W3C Amaya. PUT requests are much simpler than a file upload and they look something like this:

```
PUT /path/filename.html HTTP/1.1
```

This would normally mean that the remote client would like to save the content that follows as: /path/filename.html in your web tree. It is obviously not a good idea for Apache or PHP to automatically let everybody overwrite any files in your web tree. So, to handle such a request you have to first tell your web server that you want a certain PHP script to handle the request. In Apache you do this with the Script directive. It can be placed almost anywhere in your Apache configuration file. A common place is inside a <Directory> block or perhaps inside a <Virtualhost> block. A line like this would do the trick:

```
Script PUT /put.php
```

This tells Apache to send all PUT requests for URIs that match the context in which you put this line to the put.php script. This assumes, of course, that you have PHP enabled for the .php extension and PHP is active.

Inside your put.php file you would then do something like this:

```
<?php copy($PHP_UPLOADED_FILE_NAME,$DOCUMENT_ROOT.$REQUEST_URI); ?>
```

This would copy the file to the location requested by the remote client. You would probably want to perform some checks and/or authenticate the user before performing this file copy. The only trick here is that when PHP sees a PUT-method request it stores the uploaded file in a temporary file just like those handled but the POST-method. When the request ends, this temporary file is deleted. So, your PUT handling PHP script has to copy that file somewhere. The filename of this temporary file is in the $PHP_PUT_FILENAME variable, and you can see the suggested destination filename in the $REQUEST_URI (may vary on non-Apache web servers). This destination filename is the one that the remote client specified. You do not have to listen to this client. You could, for example, copy all uploaded files to a special uploads directory.

Chapter 19. Using remote files

As long as allow_url_fopen is enabled in php.ini, you can use HTTP and FTP URLs with most of the functions that take a filename as a parameter. In addition, URLs can be used with the **include()**, **include_once()**, **require()** and **require_once()** statements. See Appendix I for more information about the protocols supported by PHP.

> **Note:** In PHP 4.0.3 and older, in order to use URL wrappers, you were required to configure PHP using the configure option --enable-url-fopen-wrapper.

> **Note:** The Windows versions of PHP earlier than PHP 4.3 did not support remote file accessing for the following functions: **include()**, **include_once()**, **require()**, **require_once()**, and the imagecreatefromXXX functions in the Reference XLI, Image functions extension.

For example, you can use this to open a file on a remote web server, parse the output for the data you want, and then use that data in a database query, or simply to output it in a style matching the rest of your website.

Example 19-1. Getting the title of a remote page

```
<?php
$file = fopen ("http://www.example.com/", "r");
if (!$file) {
    echo "<p>Unable to open remote file.\n";
    exit;
}
while (!feof ($file)) {
    $line = fgets ($file, 1024);
    /* This only works if the title and its tags are on one line */
    if (eregi ("<title>(.*)</title>", $line, $out)) {
        $title = $out[1];
        break;
    }
}
fclose($file);
?>
```

You can also write to files on an FTP server (provided that you have connected as a user with the correct access rights). You can only create new files using this method; if you try to overwrite a file that already exists, the **fopen()** call will fail.

To connect as a user other than 'anonymous', you need to specify the username (and possibly password) within the URL, such as 'ftp://user:password@ftp.example.com/path/to/file'. (You can use the same sort of syntax to access files via HTTP when they require Basic authentication.)

Example 19-2. Storing data on a remote server

```
<?php
$file = fopen ("ftp://ftp.example.com/incoming/outputfile", "w");
if (!$file) {
    echo "<p>Unable to open remote file for writing.\n";
    exit;
}
/* Write the data here. */
fputs ($file, $_SERVER['HTTP_USER_AGENT'] . "\n");
fclose ($file);
?>
```

Note: You might get the idea from the example above that you can use this technique to write to a remote log file. Unfortunately that would not work because the **fopen()** call will fail if the remote file already exists. To do distributed logging like that, you should take a look at **syslog()**.

Chapter 20. Connection handling

Note: The following applies to 3.0.7 and later.

Internally in PHP a connection status is maintained. There are 3 possible states:

- 0 - NORMAL
- 1 - ABORTED
- 2 - TIMEOUT

When a PHP script is running normally the NORMAL state, is active. If the remote client disconnects the ABORTED state flag is turned on. A remote client disconnect is usually caused by the user hitting his STOP button. If the PHP-imposed time limit (see **set_time_limit()**) is hit, the TIMEOUT state flag is turned on.

You can decide whether or not you want a client disconnect to cause your script to be aborted. Sometimes it is handy to always have your scripts run to completion even if there is no remote browser receiving the output. The default behaviour is however for your script to be aborted when the remote client disconnects. This behaviour can be set via the ignore_user_abort php.ini directive as well as through the corresponding "php_value ignore_user_abort" Apache .conf directive or with the **ignore_user_abort()** function. If you do not tell PHP to ignore a user abort and the user aborts, your script will terminate. The one exception is if you have registered a shutdown function using **register_shutdown_function()**. With a shutdown function, when the remote user hits his STOP button, the next time your script tries to output something PHP will detect that the connection has been aborted and the shutdown function is called. This shutdown function will also get called at the end of your script terminating normally, so to do something different in case of a client disconnect you can use the **connection_aborted()** function. This function will return **TRUE** if the connection was aborted.

Your script can also be terminated by the built-in script timer. The default timeout is 30 seconds. It can be changed using the max_execution_time php.ini directive or the corresponding "php_value max_execution_time" Apache .conf directive as well as with the **set_time_limit()** function. When the timer expires the script will be aborted and as with the above client disconnect case, if a shutdown function has been registered it will be called. Within this shutdown function you can check to see if a timeout caused the shutdown function to be called by calling the **connection_timeout()** function. This function will return **TRUE** if a timeout caused the shutdown function to be called.

One thing to note is that both the ABORTED and the TIMEOUT states can be active at the same time. This is possible if you tell PHP to ignore user aborts. PHP will still note the fact that a user may have broken the connection, but the script will keep running. If it then hits the time limit it will be aborted and your shutdown function, if any, will be called. At this point you will find that **connection_timeout()** and **connection_aborted()** return **TRUE**. You can also check both states in a single call by using the **connection_status()**. This function returns a bitfield of the active states. So, if both states are active it would return 3, for example.

Chapter 21. Persistent Database Connections

Persistent connections are SQL links that do not close when the execution of your script ends. When a persistent connection is requested, PHP checks if there's already an identical persistent connection (that remained open from earlier) - and if it exists, it uses it. If it does not exist, it creates the link. An 'identical' connection is a connection that was opened to the same host, with the same username and the same password (where applicable).

Note: There are other extensions that provide persistent connections, such as the IMAP extension.

People who aren't thoroughly familiar with the way web servers work and distribute the load may mistake persistent connects for what they're not. In particular, they do not give you an ability to open 'user sessions' on the same SQL link, they do not give you an ability to build up a transaction efficiently, and they don't do a whole lot of other things. In fact, to be extremely clear about the subject, persistent connections don't give you any functionality that wasn't possible with their non-persistent brothers.

Why?

This has to do with the way web servers work. There are three ways in which your web server can utilize PHP to generate web pages.

The first method is to use PHP as a CGI "wrapper". When run this way, an instance of the PHP interpreter is created and destroyed for every page request (for a PHP page) to your web server. Because it is destroyed after every request, any resources that it acquires (such as a link to an SQL database server) are closed when it is destroyed. In this case, you do not gain anything from trying to use persistent connections -- they simply don't persist.

The second, and most popular, method is to run PHP as a module in a multiprocess web server, which currently only includes Apache. A multiprocess server typically has one process (the parent) which coordinates a set of processes (its children) who actually do the work of serving up web pages. When each request comes in from a client, it is handed off to one of the children that is not already serving another client. This means that when the same client makes a second request to the server, it may be serviced by a different child process than the first time. What a persistent connection does for you in this case it make it so each child process only needs to connect to your SQL server the first time that it serves a page that makes use of such a connection. When another page then requires a connection to the SQL server, it can reuse the connection that child established earlier.

The last method is to use PHP as a plug-in for a multithreaded web server. Currently PHP 4 has support for ISAPI, WSAPI, and NSAPI (on Windows), which all allow PHP to be used as a plug-in on multithreaded servers like Netscape FastTrack (iPlanet), Microsoft's Internet Information Server (IIS), and O'Reilly's WebSite Pro. The behavior is essentially the same as for the multiprocess model described before. Note that SAPI support is not available in PHP 3.

If persistent connections don't have any added functionality, what are they good for?

The answer here is extremely simple -- efficiency. Persistent connections are good if the overhead to create a link to your SQL server is high. Whether or not this overhead is really high depends on many factors. Like, what kind of database it is, whether or not it sits on the same computer on which your web server sits, how loaded the machine the SQL server sits on is and so forth. The bottom line is that if that connection overhead is high, persistent connections help you considerably. They cause the child process to simply connect only once for its entire lifespan, instead of every time it processes a page that requires connecting to the SQL server. This means that for every child that opened a persistent connection will have its own open persistent connection to the server. For example, if you had 20 different child processes that ran a script that made a persistent connection to your SQL server, you'd have 20 different connections to the SQL server, one from each child.

Note, however, that this can have some drawbacks if you are using a database with connection limits that are exceeded by persistent child connections. If your database has a limit of 16 simultaneous connections, and in the course of a busy server

Note, however, that this can have some drawbacks if you are using a database with connection limits that are exceeded by persistent child connections. If your database has a limit of 16 simultaneous connections, and in the course of a busy server session, 17 child threads attempt to connect, one will not be able to. If there are bugs in your scripts which do not allow the connections to shut down (such as infinite loops), the database with only 16 connections may be rapidly swamped. Check your database documentation for information on handling abandoned or idle connections.

Warning
There are a couple of additional caveats to keep in mind when using persistent connections. One is that when using table locking on a persistent connection, if the script for whatever reason cannot release the lock, then subsequent scripts using the same connection will block indefinitely and may require that you either restart the httpd server or the database server. Another is that when using transactions, a transaction block will also carry over to the next script which uses that connection if script execution ends before the transaction block does. In either case, you can use **register_shutdown_function()** to register a simple cleanup function to unlock your tables or roll back your transactions. Better yet, avoid the problem entirely by not using persistent connections in scripts which use table locks or transactions (you can still use them elsewhere).

An important summary. Persistent connections were designed to have one-to-one mapping to regular connections. That means that you should always be able to replace persistent connections with non-persistent connections, and it won't change the way your script behaves. It may (and probably will) change the efficiency of the script, but not its behavior!

See also **fbsql_pconnect()**, **ibase_pconnect()**, **ifx_pconnect()**, **imap_popen()**, **ingres_pconnect()**, **msql_pconnect()**, **mssql_pconnect()**, **mysql_pconnect()**, **OCIPLogon()**, **odbc_pconnect()**, **Ora_pLogon()**, **pfsockopen()**, **pg_pconnect()**, and **sybase_pconnect()**.

Chapter 22. Safe Mode

The PHP safe mode is an attempt to solve the shared-server security problem. It is architecturally incorrect to try to solve this problem at the PHP level, but since the alternatives at the web server and OS levels aren't very realistic, many people, especially ISP's, use safe mode for now.

Security and Safe Mode

Table 22-1. Security and Safe Mode Configuration Directives

Name	Default	Changeable
safe_mode	"0"	PHP_INI_SYSTEM
safe_mode_gid	"0"	PHP_INI_SYSTEM
safe_mode_include_dir	NULL	PHP_INI_SYSTEM
safe_mode_exec_dir	""	PHP_INI_SYSTEM
safe_mode_allowed_env_vars	PHP_	PHP_INI_SYSTEM
safe_mode_protected_env_vars	LD_LIBRARY_PATH	PHP_INI_SYSTEM
open_basedir	NULL	PHP_INI_SYSTEM
disable_functions	""	PHP_INI_SYSTEM

For further details and definition of the PHP_INI_* constants see **ini_set()**.

Here is a short explanation of the configuration directives.

safe_mode **boolean**

Whether to enable PHP's safe mode. Read the Security and chapter for more information.

safe_mode_gid **boolean**

By default, Safe Mode does a UID compare check when opening files. If you want to relax this to a GID compare, then turn on safe_mode_gid. Whether to use UID (**FALSE**) or GID (**TRUE**) checking upon file access.

safe_mode_include_dir **string**

UID/GID checks are bypassed when including files from this directory and its subdirectories (directory must also be in include_path or full path must including).

As of PHP 4.2.0, this directive can take a semi-colon separated path in a similar fashion to the include_path directive, rather than just a single directory.

The restriction specified is actually a prefix, not a directory name. This means that "safe_mode_include_dir = /dir/incl" also allows access to "/dir/include" and "/dir/incls" if they exist. When you want to restrict access to only the specified directory, end with a slash. For example: "safe_mode_include_dir = /dir/incl/"

safe_mode_exec_dir **string**

If PHP is used in safe mode, **system()** and the other functions executing system programs refuse to start programs that are not in this directory.

safe_mode_allowed_env_vars **string**

Setting certain environment variables may be a potential security breach. This directive contains a comma-delimited list of prefixes. In Safe Mode, the user may only alter environment variables whose names begin with the prefixes supplied here. By default, users will only be able to set environment variables that begin with PHP_ (e.g. PHP_FOO=BAR).

Note: If this directive is empty, PHP will let the user modify ANY environment variable!

safe_mode_protected_env_vars **string**

This directive contains a comma-delimited list of environment variables that the end user won't be able to change using **putenv()**. These variables will be protected even if safe_mode_allowed_env_vars is set to allow to change them.

open_basedir **string**

Limit the files that can be opened by PHP to the specified directory-tree, including the file itself. This directive is NOT affected by whether Safe Mode is turned On or Off.

When a script tries to open a file with, for example, fopen or gzopen, the location of the file is checked. When the file is outside the specified directory-tree, PHP will refuse to open it. All symbolic links are resolved, so it's not possible to avoid this restriction with a symlink.

The special value . indicates that the directory in which the script is stored will be used as base-directory.

Under Windows, separate the directories with a semicolon. On all other systems, separate the directories with a colon. As

The special value . indicates that the directory in which the script is stored will be used as base directory.

Under Windows, separate the directories with a semicolon. On all other systems, separate the directories with a colon. As an Apache module, open_basedir paths from parent directories are now automatically inherited.

The restriction specified with open_basedir is actually a prefix, not a directory name. This means that "open_basedir = /dir/incl" also allows access to "/dir/include" and "/dir/incls" if they exist. When you want to restrict access to only the specified directory, end with a slash. For example: "open_basedir = /dir/incl/"

Note: Support for multiple directories was added in 3.0.7.

The default is to allow all files to be opened.

disable_functions **string**

This directive allows you to disable certain functions for security reasons. It takes on a comma-dilimited list of function names. disable_functions is not affected by Safe Mode.

This directive must be set in php.ini For example, you cannot set this in httpd.conf.

See also: register_globals, display_errors, and log_errors

When safe_mode is on, PHP checks to see if the owner of the current script matches the owner of the file to be operated on by a file function. For example:

```
-rw-rw-r--   1 rasmus   rasmus      33 Jul  1 19:20 script.php
-rw-r--r--   1 root     root      1116 May 26 18:01 /etc/passwd
```

Running this script.php

```
<?php
 readfile('/etc/passwd');
?>
```

results in this error when safe mode is enabled:

```
Warning: SAFE MODE Restriction in effect. The script whose uid is 500 is not
allowed to access /etc/passwd owned by uid 0 in /docroot/script.php on line 2
```

However, there may be environments where a strict UID check is not appropriate and a relaxed GID check is sufficient. This is supported by means of the safe_mode_gid switch. Setting it to On performs the relaxed GID checking, setting it to Off (the default) performs UID checking.

If instead of safe_mode, you set an open_basedir directory then all file operations will be limited to files under the specified directory For example (Apache httpd.conf example):

```
<Directory /docroot>
  php_admin_value open_basedir /docroot
</Directory>
```

If you run the same script.php with this open_basedir setting then this is the result:

```
Warning: open_basedir restriction in effect. File is in wrong directory in
/docroot/script.php on line 2
```

You can also disable individual functions. Note that the disable_functions directive can not be used outside of the php.ini file which means that you cannot disable functions on a per-virtualhost or per-directory basis in your httpd.conf file. If we add this to our php.ini file:

```
disable_functions readfile,system
```

Then we get this output:

```
Warning: readfile() has been disabled for security reasons in
/docroot/script.php on line 2
```

Functions restricted/disabled by safe mode

This is a still probably incomplete and possibly incorrect listing of the functions limited by safe mode.

Table 22-2. Safe mode limited functions

Function	Limitations
dbmopen()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
dbase_open()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
filepro()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
filepro_rowcount()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
filepro_retrieve()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
ifx_*()	sql_safe_mode restrictions, (!= safe mode)
ingres_*()	sql_safe_mode restrictions, (!= safe mode)
mysql_*()	sql_safe_mode restrictions, (!= safe mode)
pg_loimport()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
posix_mkfifo()	Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed.
putenv()	Obeys the safe_mode_protected_env_vars and safe_mode_allowed_env_vars ini-directives. See also the

putenv()	Obeys the safe_mode_protected_env_vars and safe_mode_allowed_env_vars ini-directives. See also the documentation on **putenv()**
move_uploaded_file()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
chdir()	Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed.
dl()	This function is disabled in safe mode.
backtick operator	This function is disabled in safe mode.
shell_exec() (functional equivalent of backticks)	This function is disabled in safe mode.
exec()	You can only execute executables within the safe_mode_exec_dir. For practical reasons it's currently not allowed to have .. components in the path to the executable.
system()	You can only execute executables within the safe_mode_exec_dir. For practical reasons it's currently not allowed to have .. components in the path to the executable.
passthru()	You can only execute executables within the safe_mode_exec_dir. For practical reasons it's currently not allowed to have .. components in the path to the executable.
popen()	You can only execute executables within the safe_mode_exec_dir. For practical reasons it's currently not allowed to have .. components in the path to the executable.
mkdir()	Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed.
rmdir()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
rename()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed.
unlink()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed.
copy()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed. (on source and target)
chgrp()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
chown()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed.
chmod()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. In addition, you cannot set the SUID, SGID and sticky bits
touch()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed.
symlink()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed. (note: only the target is checked)
link()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed. (note: only the target is checked)
getallheaders()	In safe mode, headers beginning with 'authorization' (case-insensitive) will not be returned. Warning: this is broken with the aol-server implementation of **getallheaders()**!
header()	In safe mode, the uid of the script is added to the realm part of the WWW-Authenticate header if you set this header (used for HTTP Authentication).
PHP_AUTH variables	In safe mode, the variables PHP_AUTH_USER, PHP_AUTH_PW, and PHP_AUTH_TYPE are not available in $_SERVER. Regardless, you can still use REMOTE_USER for the USER. (note: only affected since PHP 4.3.0)
highlight_file(), **show_source()**	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed. (note: only affected since PHP 4.2.1)
parse_ini_file()	Checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being

parse_ini_file()	on have the same UID (owner) as the script that is being executed. Checks whether the directory in which you are about to operate has the same UID (owner) as the script that is being executed. (note: only affected since PHP 4.2.1)
Any function that uses php4/main/fopen_wrappers.c	??

Chapter 23. Using PHP from the command line

As of version 4.3.0, PHP supports a new SAPI type (Server Application Programming Interface) named CLI which means Command Line Interface. As the name implies, this SAPI type main focus is on developing shell (or desktop as well) applications with PHP. There are quite a few differences between the CLI SAPI and other SAPIs which are explained in this chapter. It's worth mentioning that CLI and CGI are different SAPI's although they do share many of the same behaviors.

The CLI SAPI was released for the first time with PHP 4.2.0, but was still experimental and had to be explicitly enabled with --enable-cli when running ./configure. Since PHP 4.3.0 the CLI SAPI is no longer experimental and the option --enable-cli is on by default. You may use --disable-cli to disable it.

As of PHP 4.3.0, the name, location and existence of the CLI/CGI binaries will differ depending on how PHP is installed on your system. By default when executing make, both the CGI and CLI are built and placed as sapi/cgi/php and sapi/cli/php respectively, in your php source directory. You will note that both are named php. What happens during make install depends on your configure line. If a module SAPI is chosen during configure, such as apxs, or the --disable-cgi option is used, the CLI is copied to {PREFIX}/bin/php during make install otherwise the CGI is placed there. So, for example, if --with--apxs is in your configure line then the CLI is copied to {PREFIX}/bin/php during make install. If you want to override the installation of the CGI binary, use make install-cli after make install. Alternatively you can specify --disable-cgi in your configure line.

> **Note:** Because both --enable-cli and --enable-cgi are enabled by default, simply having --enable-cli in your configure line does not necessarily mean the CLI will be copied as {PREFIX}/bin/php during make install.

The windows package distributes the CGI as php.exe and has a folder named cli with the CLI in it, so: cli/php.exe.

> **What SAPI do I have?:** From a shell, typing php -v will tell you whether php is CGI or CLI. See also the function **php_sapi_name()** and the constant **PHP_SAPI**.

Remarkable differences of the CLI SAPI compared to other SAPIs:

- Unlike the CGI SAPI, no headers are written to the output.
  Though the CGI SAPI provides a way to suppress HTTP headers, there's no equivalent switch to enable them in the CLI SAPI.
  CLI is started up in quiet mode by default, though the -q switch is kept for compatibility so that you can use older CGI scripts.
  It does not change the working directory to that of the script. (-C switch kept for compatibility)
  Plain text error messages (no HTML formatting).
- There are certain php.ini directives which are overriden by the CLI SAPI because they do not make sense in shell environments:

  Table 23-1. Overriden php.ini directives

Directive	CLI SAPI default value	Comment
html_errors	**FALSE**	It can be quite hard to read the error message in your shell when it's cluttered with all those meaningless HTML tags, therefore this directive defaults to **FALSE**.
implicit_flush	**TRUE**	It is desired that any output coming from **print()**, **echo()** and friends is immediately written to the output and not cached in any buffer. You still can use output buffering if you want to defer or manipulate standard output.
max_execution_time	0 (unlimited)	Due to endless possibilities of using PHP in shell environments, the maximum execution time has been set to unlimited. Whereas applications written for the web are often executed very quickly, shell application tend to have a much longer execution time.
register_argc_argv	**TRUE**	Because this setting is **TRUE** you will always have access to argc (number of arguments passed to the application) and argv (array of the actual arguments) in the CLI SAPI. As of PHP 4.3.0, the PHP variables $argc and $argv are registered and filled in with the appropriate values when using the CLI SAPI. Prior to this version, the creation of these variables behaved as they do in CGI and MODULE versions which requires the PHP directive register_globals to be on. Regardless of version or register_globals setting, you can always go through either $_SERVER or $HTTP_SERVER_VARS. Example: $_SERVER['argv']

  > **Note:** These directives cannot be initialized with another value from the configuration file php.ini or a custom one (if specified). This is a limitation because those default values are applied after all configuration files have been parsed. However, their value can be changed during runtime (which does not make sense for all of those directives, e.g. register_argc_argv).
- To ease working in the shell environment, the following constants are defined:

  Table 23-2. CLI specific Constants

Constant	Description
STDIN	An already opened stream to stdin. This saves opening it with `$stdin = fopen('php://stdin', 'r');`
STDOUT	An already opened stream to stdout. This saves opening it with `$stdout = fopen('php://stdout', 'w');`

STDOUT	An already opened stream to stdout. This saves opening it with `$stdout = fopen('php://stdout', 'w');`
STDERR	An already opened stream to stderr. This saves opening it with `$stderr = fopen('php://stderr', 'w');`

Given the above, you don't need to open e.g. a stream for stderr yourself but simply use the constant instead of the stream resource:

```
php -r 'fwrite(STDERR, "stderr\n");'
```

You do not need to explicitly close these streams, as they are closed automatically by PHP when your script ends.

- The CLI SAPI does **not** change the current directory to the directory of the executed script!
  Example showing the difference to the CGI SAPI:

```
<?php
    /* Our simple test application named test.php*/
    echo getcwd(), "\n";
?>
```

When using the CGI version, the output is:

```
$ pwd
/tmp

$ php -q another_directory/test.php
/tmp/another_directory
```

This clearly shows that PHP changes its current directory to the one of the executed script.
Using the CLI SAPI yields:

```
$ pwd
/tmp

$ php -f another_directory/test.php
/tmp
```

This allows greater flexibility when writing shell tools in PHP.

> **Note:** The CGI SAPI supports the CLI SAPI behaviour by means of the -C switch when run from the command line.

The list of command line options provided by the PHP binary can be queried anytime by running PHP with the -h switch:

```
Usage: php [options] [-f] <file> [args...]
       php [options] -r <code> [args...]
       php [options] [-- args...]
  -s               Display colour syntax highlighted source.
  -w               Display source with stripped comments and whitespace.
  -f <file>        Parse <file>.
  -v               Version number
  -c <path>|<file> Look for php.ini file in this directory
  -a               Run interactively
  -d foo[=bar]     Define INI entry foo with value 'bar'
  -e               Generate extended information for debugger/profiler
  -z <file>        Load Zend extension <file>.
  -l               Syntax check only (lint)
  -m               Show compiled in modules
  -i               PHP information
  -r <code>        Run PHP <code> without using script tags <?..?>
  -h               This help

  args...          Arguments passed to script. Use -- args when first argument
                   starts with - or script is read from stdin
```

The CLI SAPI has three different ways of getting the PHP code you want to execute:

1. Telling PHP to execute a certain file.

   ```
   php my_script.php

   php -f my_script.php
   ```

   Both ways (whether using the -f switch or not) execute the file my_script.php. You can choose any file to execute - your PHP scripts do not have to end with the .php extension but can have any name or extension you wish.
2. Pass the PHP code to execute directly on the command line.

   ```
   php -r 'print_r(get_defined_constants());'
   ```

   Special care has to be taken in regards of shell variable substitution and quoting usage.

   > **Note:** Read the example carefully, there are no beginning or ending tags! The -r switch simply does not need them. Using them will lead to a parser error.
3. Provide the PHP code to execute via standard input (stdin).
   This gives the powerful ability to dynamically create PHP code and feed it to the binary, as shown in this (fictional) example:

   ```
   $ some_application | some_filter | php | sort -u >final_output.txt
   ```

You cannot combine any of the three ways to execute code.

Like every shell application, the PHP binary accepts a number of arguments but your PHP script can also receive arguments. The number of arguments which can be passed to your script is not limited by PHP (the shell has a certain size limit in the number of characters which can be passed: usually you won't hit this limit). The arguments passed to your script are available in the global array $argv. The zero index always contains the script name (which is - in case the PHP code is coming from either standard input or from the command line switch -r). The second registered global variable is $argc which contains the number of elements in the $argv array (**not** the number of arguments passed to the script).

As long as the arguments you want to pass to your script do not start with the - character, there's nothing special to watch out for. Passing an argument to your script which starts with a - will cause trouble because PHP itself thinks it has to handle it. To prevent this, use the argument list separator --. After this separator has been parsed by PHP, every argument following it is passed untouched to your script.

```
# This will not execute the given code but will show the PHP usage
# The -- separator is not required
$ php -r -- -h
# This will pass the -h argument to the PHP code
$ php -r 'var_dump($argv);' -h
```

passed untouched to your script.

```
# This will not execute the given code but will show the PHP usage
$ php -r 'var_dump($argv);' -h
Usage: php [options] [-f] <file> [args...]
[...]

# This will pass the '-h' argument to your script and prevent PHP from showing it's usage
$ php -r 'var_dump($argv);' -- -h
array(2) {
  [0]=>
  string(1) "-"
  [1]=>
  string(2) "-h"
}
```

However, there's another way of using PHP for shell scripting. You can write a script where the first line starts with #!/usr/bin/php. Following this you can place normal PHP code included within the PHP starting and end tags. Once you have set the execution attributes of the file appropriately (e.g. chmod +x test) your script can be executed like a normal shell or perl script:

```
#!/usr/bin/php
<?php
    var_dump($argv);
?>
```

Assuming this file is named test in the current directory, we can now do the following:

```
$ chmod 755 test
$ ./test -h -- foo
array(4) {
  [0]=>
  string(6) "./test"
  [1]=>
  string(2) "-h"
  [2]=>
  string(2) "--"
  [3]=>
  string(3) "foo"
}
```

As you see, in this case no care needs to be taken when passing parameters which start with - to your script.

Table 23-3. Command line options

Option	Description
-s	Display colour syntax highlighted source. This option uses the internal mechanism to parse the file and produces a HTML highlighted version of it and writes it to standard output. Note that all it does it to generate a block of <code> [...] </code> HTML tags, no HTML headers. **Note:** This option does not work together with the -r option.
-w	Display source with stripped comments and whitespace. **Note:** This option does not work together with the -r option.
-f	Parses and executed the given filename to the -f option. This switch is optional and can be left out. Only providing the filename to execute is sufficient.

-v	Writes the PHP, PHP SAPI, and Zend version to standard output, e.g.
	
	`$ php -v`
	`PHP 4.3.0 (cli), Copyright (c) 1997-2002 The PHP Group`
	`Zend Engine v1.3.0, Copyright (c) 1998-2002 Zend Technologies`

-c	With this option one can either specify a directory where to look for php.ini or you can specify a custom INI file directly (which does not need to be named php.ini), e.g.:
	
	`$ php -c /custom/directory/ my_script.php`
	
	`$ php -c /custom/directory/custom-file.ini my_script.php`

-a	Runs PHP interactively.

-d	This option allows you to set a custom value for any of the configuration directives allowed in php.ini. The syntax is:
	
	`-d configuration_directive[=value]`
	
	Examples:
	
	`# Omitting the value part will set the given configuration directive to "1"`
	`$ php -d max_execution_time -r '$foo = ini_get("max_execution_time"); var_dump($foo);'`
	`string(1) "1"`
	
	`# Passing an empty value part will set the configuration directive to ""`
	`php -d max_execution_time= -r '$foo = ini_get("max_execution_time"); var_dump($foo);'`
	`string(0) ""`
	
	`# The configuration directive will be set to anything passed after the '=' character`
	`$ php -d max_execution_time=20 -r '$foo = ini_get("max_execution_time"); var_dump($foo);'`
	`string(2) "20"`
	`$ php -d max_execution_time=doesntmakesense -r '$foo = ini_get("max_execution_time"); var_dump($foo);'`
	`string(15) "doesntmakesense"`

-e	Generate extended information for debugger/profiler.
-z	Load Zend extension. If only a filename is given, PHP tries to load this extension from the current default library path on your system (usually specified /etc/ld.so.conf on Linux systems). Passing a filename with an absolute path information will not use the systems library search path. A relative filename with a directory information will tell PHP only to try to load the extension relative to the current directory.
-l	This option provides a convenient way to only perform a syntax check on the given PHP code. On succes, the text No syntax errors detected in <filename> is written to

	directory information will tell PHP only to try to load the extension relative to the current directory.
-l	This option provides a convenient way to only perform a syntax check on the given PHP code. On succes, the text No syntax errors detected in <filename> is written to standard output and the shell return code is 0. On failure, the text Errors parsing <filename> in addition to the internal parser error message is written to standard output and the shell return code is set to 255. This option won't find fatal errors (like undefined functions). Use -f if you would like to test for fatal errors too. **Note:** This option does not work together with the -r option.
-m	Using this option, PHP prints out the built in (and loaded) PHP and Zend modules: $ php -m [PHP Modules] xml tokenizer standard session posix pcre overload mysql mbstring ctype [Zend Modules]
-i	This command line option calls **phpinfo()**, and prints out the results. If PHP is not working correctly, it is advisable to use php -i and see whether any error messages are printed out before or in place of the information tables. Beware that the output is in HTML and therefore quite huge.

-r	This option allows execution of PHP right from within the command line. The PHP start and end tags (<?php and ?>) are **not needed** and will cause a parser error if present.
	Note: Care has to be taken when using this form of PHP to not collide with command line variable substitution done by the shell.
	Example showing a parser error
	$ php -r "$foo = get_defined_constants();"
	Command line code(1) : Parse error - parse error, unexpected '='
	The problem here is that the sh/bash performs variable substitution even when using double quotes ". Since the variable $foo is unlikely to be defined, it expands to nothing which results in the code passed to PHP for execution actually reading:
	$ php -r " = get_defined_constants();"
	The correct way would be to use single quotes '. Variables in single-quoted strings are not expanded by sh/bash.
	$ php -r '$foo = get_defined_constants(); var_dump($foo);'
	array(370) {
	["E_ERROR"]=>
	int(1)
	["E_WARNING"]=>
	int(2)
	["E_PARSE"]=>
	int(4)
	["E_NOTICE"]=>
	int(8)
	["E_CORE_ERROR"]=>
	[...]
	If you are using a shell different from sh/bash, you might experience further issues. Feel free to open a bug report or send a mail to phpdoc@lists.php.net. One can still easily run into troubles when trying to get shell variables into the code or using backslashes for escaping. You've been warned.
	Note: -r is available in the CLI SAPI and not in the CGI SAPI.

-h	With this option, you can get information about the actual list of command line options and some one line descriptions about what they do.

The PHP executable can be used to run PHP scripts absolutely independent from the web server. If you are on a Unix system, you should add a special first line to your PHP script, and make it executable, so the system will know, what program should run the script. On a Windows platform you can associate php.exe with the double click option of the .php files, or you can make a batch file to run the script through PHP. The first line added to the script to work on Unix won't hurt on Windows, so you can write cross platform programs this way. A simple example of writing a command line PHP program can be found below.

Example 23-1. Script intended to be run from command line (script.php)

```
#!/usr/bin/php
<?php

if ($argc != 2 || in_array($argv[1], array('--help', '-help', '-h', '-?'))) {
?>

This is a command line PHP script with one option.

  Usage:
  <?php echo $argv[0]; ?> <option>

  <option> can be some word you would like
  to print out. With the --help, -help, -h,
  or -? options, you can get this help.

<?php
} else {
    echo $argv[1];
}
?>
```

In the script above, we used the special first line to indicate that this file should be run by PHP. We work with a CLI version here, so there will be no HTTP header printouts. There are two variables you can use while writing command line applications with PHP: $argc and $argv. The first is the number of arguments plus one (the name of the script running). The second is an array

here, so there will be no HTTP header printouts. There are two variables you can use while writing command line applications with PHP: $argc and $argv. The first is the number of arguments plus one (the name of the script running). The second is an array containing the arguments, starting with the script name as number zero ($argv[0]).

In the program above we checked if there are less or more than one arguments. Also if the argument was --help, -help, -h or -?, we printed out the help message, printing the script name dynamically. If we received some other argument we echoed that out.

If you would like to run the above script on Unix, you need to make it executable, and simply call it as script.php echothis or script.php -h. On Windows, you can make a batch file for this task:

Example 23-2. Batch file to run a command line PHP script (script.bat)

```
@c:\php\cli\php.exe script.php %1 %2 %3 %4
```

Assuming you named the above program script.php, and you have your CLI php.exe in c:\php\cli\php.exe this batch file will run it for you with your added options: script.bat echothis or script.bat -h.

See also the Readline extension documentation for more functions you can use to enhance your command line applications in PHP.

IV. Function Reference

Table of Contents

I. Apache-specific Functions

Introduction

These functions are only available when running PHP as an Apache 1.x module.

Installation

For PHP installation on Apache 1.x see the Apache section in the installation chapter.

Runtime Configuration

The behaviour of the Apache PHP module is affected by settings in php.ini. Configuration settings from php.ini may be overridden by php_flag settings in the server configuration file or local .htaccess files.

Example 1. Turning off PHP parsing for a directory using .htaccess

```
php_flag engine off
```

Table 1. Apache configuration options

Name	Default	Changeable	Function
engine	On	PHP_INI_ALL	turns PHP parsing on or off
child_terminate	Off	PHP_INI_ALL	specify whether PHP scripts may request child process termination on end of request, see also **apache_child_terminate()**
last_modified	Off	PHP_INI_ALL	send PHP scripts modification date as Last-Modified: header for this request
xbit_hack	Off	PHP_INI_ALL	parse files with executable bit set as PHP regardles of their file ending

Here is a short explanation of the configuration directives.

engine **boolean**

> This directive is really only useful in the Apache module version of PHP. It is used by sites that would like to turn PHP parsing on and off on a per-directory or per-virtual server basis. By putting **engine off** in the appropriate places in the httpd.conf file, PHP can be enabled or disabled.

Resource Types

This extension has no resource types defined.

Predefined Constants

Predefined Constants

This extension has no constants defined.

Table of Contents

apache_child_terminate

(PHP 4 >= 4.0.5)

apache_child_terminate -- Terminate apache process after this request

Description

bool **apache_child_terminate** (void)

apache_child_terminate() will register the Apache process executing the current PHP request for termination once execution of PHP code it is completed. It may be used to terminate a process after a script with high memory consumption has been run as memory will usually only be freed internally but not given back to the operating system.

> **Note:** The availability of this feature is controlled by the php.ini directive apache.child_terminate, which is set to off by default.
>
> This feature is also not available on multithreaded versions of apache like the win32 version.

See also **exit()**.

apache_lookup_uri

(PHP 3>= 3.0.4, PHP 4)

apache_lookup_uri -- Perform a partial request for the specified URI and return all info about it

Description

object **apache_lookup_uri** (string filename)

This performs a partial request for a URI. It goes just far enough to obtain all the important information about the given resource and returns this information in a class. The properties of the returned class are:

status
the_request
status_line
method
content_type
handler
uri
filename
path_info
args
boundary
no_cache
no_local_copy
allowed
send_bodyct
bytes_sent
byterange
clength
unparsed_uri
mtime
request_time

> **Note:** **apache_lookup_uri()** only works when PHP is installed as an Apache module.

apache_note

(PHP 3>= 3.0.2, PHP 4)

apache_note -- Get and set apache request notes

Description

string **apache_note** (string note_name [, string note_value])

apache_note() is an Apache-specific function which gets and sets values in a request's notes table. If called with one argument, it returns the current value of note note_name. If called with two arguments, it sets the value of note note_name to note_value

apache_note() is an Apache-specific function which gets and sets values in a request's notes table. If called with one argument, it returns the current value of note note_name. If called with two arguments, it sets the value of note note_name to note_value and returns the previous value of note note_name.

apache_request_headers

(PHP 4 >= 4.3.0)

apache_request_headers -- Fetch all HTTP request headers

Description

array **apache_request_headers** (void)

apache_request_headers() returns an associative array of all the HTTP headers in the current request. This is only supported when PHP runs as an Apache module.

> **Note:** Prior to PHP 4.3.0, **apache_request_headers()** was called **getallheaders()**. After PHP 4.3.0, **getallheaders()** is an alias for **apache_request_headers()**.

Example 1. apache_request_headers() Example

```
<?php
$headers = apache_request_headers();

foreach ($headers as $header => $value) {
    echo "$header: $value <br />\n";
}
?>
```

> **Note:** You can also get at the value of the common CGI variables by reading them from the environment, which works whether or not you are using PHP as an Apache module. Use **phpinfo()** to see a list of all of the available environment variables.

apache_response_headers

(PHP 4 >= 4.3.0)

apache_response_headers -- Fetch all HTTP response headers

Description

array **apache_response_headers** (void)

Returns an array of all Apache response headers. This functionality is only available in PHP version 4.3.0 and greater.

See also **getallheaders()** and **headers_sent()**.

apache_setenv

(PHP 4 >= 4.2.0)

apache_setenv -- Set an Apache subprocess_env variable

Description

int **apache_setenv** (string variable, string value [, bool walk_to_top])

Warning
This function is currently not documented; only the argument list is available.

ascii2ebcdic

(PHP 3>= 3.0.17)

ascii2ebcdic -- Translate string from ASCII to EBCDIC

Description

int **ascii2ebcdic** (string ascii_str)

ascii2ebcdic() is an Apache-specific function which is available only on EBCDIC based operating systems (OS/390, BS2000). It translates the ASCII encoded string ascii_str to its equivalent EBCDIC representation (binary safe), and returns the result.

See also the reverse function **ebcdic2ascii()**

ebcdic2ascii

(PHP 3>= 3.0.17)

ebcdic2ascii -- Translate string from EBCDIC to ASCII

Description

Description

int **ebcdic2ascii** (string ebcdic_str)

ebcdic2ascii() is an Apache-specific function which is available only on EBCDIC based operating systems (OS/390, BS2000). It translates the EBCDIC encoded string ebcdic_str to its equivalent ASCII representation (binary safe), and returns the result.

See also the reverse function **ascii2ebcdic()**

getallheaders

(PHP 3, PHP 4)

getallheaders -- Fetch all HTTP request headers

Description

array **getallheaders** (void)

getallheaders() is an alias for **apache_request_headers()**. It will return an associative array of all the HTTP headers in the current request. Please read the **apache_request_headers()** documentation for more information on how this function works.

> **Note:** In PHP 4.3.0, **getallheaders()** became an alias for **apache_request_headers()**. Essentially, it was renamed. This is because this function only works when PHP is compiled as an Apache Module.

See also **apache_request_headers()**.

virtual

(PHP 3, PHP 4)

virtual -- Perform an Apache sub-request

Description

int **virtual** (string filename)

virtual() is an Apache-specific function which is equivalent to <!--#include virtual...--> in mod_include. It performs an Apache sub-request. It is useful for including CGI scripts or .shtml files, or anything else that you would parse through Apache. Note that for a CGI script, the script must generate valid CGI headers. At the minimum that means it must generate a Content-type header. For PHP files, you need to use **include()** or **require()**; **virtual()** cannot be used to include a document which is itself a PHP file.

To run the sub-request, all buffers are terminated and flushed to the browser, pending headers are sent too.

II. Array Functions

Introduction

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables.

Simple and multi-dimensional arrays are supported, and may be either user created or created by another function. There are specific database handling functions for populating arrays from database queries, and several functions return arrays.

Please see the Arrays section of the manual for a detailed explanation of how arrays are implemented and used in PHP.

Requirements

No external libraries are needed to build this extension.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

Runtime Configuration

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

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are always available as part of the PHP core.

CASE_LOWER (integer)

> **CASE_LOWER** is used with **array_change_key_case()** and is used to convert array keys to lower case. This is also the default case for **array_change_key_case()**.

CASE_UPPER (integer)

> **CASE_UPPER** is used with **array_change_key_case()** and is used to convert array keys to upper case.

Sorting order flags:

CASE_UPPER is used with **array_change_key_case()** and is used to convert array keys to upper case.

Sorting order flags:

SORT_ASC (integer)

SORT_ASC is used with **array_multisort()** to sort in ascending order.

SORT_DESC (integer)

SORT_DESC is used with **array_multisort()** to sort in descending order.

Sorting type flags: used by various sort functions

SORT_REGULAR (integer)

SORT_REGULAR is used to compare items normally.

SORT_NUMERIC (integer)

SORT_NUMERIC is used to compare items numerically.

SORT_STRING (integer)

SORT_STRING is used to compare items as strings.

See Also

See also i**s_array()**, **explode()**, **implode()**, **split()**, **preg_split()**, and **join()**.

Table of Contents

array_change_key_case

(PHP 4 >= 4.2.0)

array_change_key_case -- Returns an array with all string keys lowercased or uppercased

Description

array **array_change_key_case** (array input [, int case])

Description

array **array_change_key_case** (array input [, int case])

array_change_key_case() changes the keys in the input array to be all lowercase or uppercase. The change depends on the last optional case parameter. You can pass two constants there, **CASE_UPPER** and **CASE_LOWER**. The default is **CASE_LOWER**. The function will leave number indices as is.

Example 1. array_change_key_case() example

```
$input_array = array("FirSt" => 1, "SecOnd" => 4);
print_r(array_change_key_case($input_array, CASE_UPPER));
```

The printout of the above program will be:

```
Array
(
    [FIRST] => 1
    [SECOND] => 2
)
```

array_chunk

(PHP 4 >= 4.2.0)

array_chunk -- Split an array into chunks

Description

array **array_chunk** (array input, int size [, bool preserve_keys])

array_chunk() splits the array into several arrays with size values in them. You may also have an array with less values at the end. You get the arrays as members of a multidimensional array indexed with numbers starting from zero.

By setting the optional preserve_keys parameter to **TRUE**, you can force PHP to preserve the original keys from the input array. If you specify **FALSE** new number indices will be used in each resulting array with indices starting from zero. The default is **FALSE**.

Example 1. array_chunk() example

```
$input_array = array('a', 'b', 'c', 'd', 'e');
print_r(array_chunk($input_array, 2));
print_r(array_chunk($input_array, 2, TRUE));
```

The printout of the above program will be:

```
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )

    [1] => Array
        (
            [0] => c
            [1] => d
        )

    [2] => Array
        (
            [0] => e
        )

)
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )

    [1] => Array
        (
            [2] => c
            [3] => d
        )

    [2] => Array
        (
            [4] => e
        )

)
```

array_count_values

(PHP 4)

(PHP 4)

array_count_values -- Counts all the values of an array

Description

array **array_count_values** (array input)

array_count_values() returns an array using the values of the input array as keys and their frequency in input as values.

Example 1. array_count_values() example

```
$array = array (1, "hello", 1, "world", "hello");
print_r(array_count_values ($array));
```

The printout of the above program will be:

```
Array
(
    [1] => 2
    [hello] => 2
    [world] => 1
)
```

array_diff_assoc

(PHP 4 >= 4.3.0)

array_diff_assoc -- Computes the difference of arrays with additional index check

Description

array **array_diff_assoc** (array array1, array array2 [, array ...])

array_diff_assoc() returns an **array** containing all the values from array1 that are not present in any of the other arguments. Note that the keys are used in the comparison unlike **array_diff()**.

Example 1. array_diff_assoc() example

```
<?php
$array1 = array ("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array ("a" => "green", "yellow", "red");
$result = array_diff_assoc ($array1, $array2);

/* The result is:
Array
(
    [b] => brown
    [c] => blue
    [0] => red
)
*/
?>
```

In our example above you see the "a" => "green" pair is present in both arrays and thus it is not in the ouput from the function. Unlike this, the pair 0 => "red" is in the ouput because in the second argument "red" has key which is 1.

Two values from key => value pairs are considered equal only if (string) $elem1 === (string) $elem2 . In other words a strict check takes place so the string representations must be the same.

> **Note:** Please note that this function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using, for example, array_diff_assoc($array1[0], $array2[0]);.

See also **array_diff()**, **array_intersect()**, and **array_intersect_assoc()**.

array_diff

(PHP 4 >= 4.0.1)

array_diff -- Computes the difference of arrays

Description

array **array_diff** (array array1, array array2 [, array ...])

array_diff() returns an array containing all the values of array1 that are not present in any of the other arguments. Note that keys are preserved.

Example 1. array_diff() example

```
$array1 = array ("a" => "green", "red", "blue", "red");
$array2 = array ("b" => "green", "yellow", "red");
$result = array_diff ($array1, $array2);
```

This makes $result have array ("blue");. Multiple occurrences in $array1 are all treated the same way.

> **Note:** Two elements are considered equal if and only if (string) $elem1 === (string) $elem2. In words: when the string representation is the same.

Note: Two elements are considered equal if and only if (string) $elem1 === (string) $elem2. In words: when the string representation is the same.

Note: Please note that this function only checks one dimension of a n-dimensional array. Of course you can check deeper dimensions by using array_diff($array1[0], $array2[0]);.

Warning
This was broken in PHP 4.0.4!

See also **array_diff_assoc()**, **array_intersect()** and **array_intersect_assoc()**.

array_fill

(PHP 4 >= 4.2.0)

array_fill -- Fill an array with values

Description

array **array_fill** (int start_index, int num, mixed value)

array_fill() fills an array with num entries of the value of the value parameter, keys starting at the start_index parameter.

Example 1. array_fill() example

```
$a = array_fill(5, 6, 'banana');
```

$a now has the following entries using **print_r()**:

```
Array
(
    [5] => banana
    [6] => banana
    [7] => banana
    [8] => banana
    [9] => banana
    [10] => banana
)
```

array_filter

(PHP 4 >= 4.0.6)

array_filter -- Filters elements of an array using a callback function

Description

array **array_filter** (array input [, callback function])

array_filter() returns an array containing all the elements of input filtered according a callback function. If the input is an associative array the keys are preserved.

Example 1. array_filter() example

```
function odd($var) {
    return ($var % 2 == 1);
}

function even($var) {
    return ($var % 2 == 0);
}

$array1 = array ("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);
$array2 = array (6, 7, 8, 9, 10, 11, 12);

echo "Odd :\n";
print_r(array_filter($array1, "odd"));
echo "Even:\n";
print_r(array_filter($array2, "even"));
```

The printout of the program above will be:

```
Odd :
Array
(
    [a] => 1
    [c] => 3
    [e] => 5
)
Even:
Array
(
    [0] => 6
    [2] => 8
    [4] => 10
    [6] => 12
)
```

```
)
```

Users may not change the array itself from the callback function. e.g. Add/delete an element, unset the array that **array_filter()** is applied to. If the array is changed, the behavior of this function is undefined.

See also **array_map()** and **array_reduce()**.

array_flip

(PHP 4)

array_flip -- Exchanges all keys with their associated values in an array

Description

array **array_flip** (array trans)

array_flip() returns an **array** in flip order, i.e. keys from trans become values and trans's values become keys.

Note that the values of trans need to be valid keys, i.e. they need to be either **integer** or **string**. A warning will be emitted if a value has the wrong type, and the key/value pair in question will not be flipped.

If a value has several occurrences, the latest key will be used as its values, and all others will be lost.

array_flip() returns **FALSE** if it fails.

Example 1. array_flip() example

```
$trans = array_flip ($trans);
$original = strtr ($str, $trans);
```

Example 2. array_flip() example : collision

```
$trans = array ("a" => 1, "b" => 1, "c" => 2);
$trans = array_flip ($trans);
print_r($trans);
```

now $trans is :

```
Array
(
    [1] => b
    [2] => c
)
```

array_intersect_assoc

(PHP 4 >= 4.3.0)

array_intersect_assoc -- Computes the intersection of arrays with additional index check

Description

array **array_intersect_assoc** (array array1, array array2 [, array ...])

array_intersect_assoc() returns an array containing all the values of array1 that are present in all the arguments. Note that the keys are used in the comparison unlike in **array_intersect()**.

Example 1. array_intersect_assoc() example

```
<?php
$array1 = array ("a" => "green", "b" => "brown", "c" => "blue", "red");
$array2 = array ("a" => "green", "yellow", "red");
$result_array = array_intersect_assoc ($array1, $array2);

/* $result_array will look like:

Array
(
    [a] => green
)

*/
?>
```

In our example you see that only the pair "a" => "green" is present in both arrays and thus is returned. The value "red" is not returned because in $array1 it's key is 2 while the key of "red" in $array2 it is 1.

The two values from the key => value pairs are considered equal only if (string) $elem1 === (string) $elem2 . In otherwords a strict type check is executed so the string representation must be the same.

See also **array_intersect()**, **array_diff()** and **array_diff_assoc()**.

array_intersect

(PHP 4 >= 4.0.1)

array_intersect -- Computes the intersection of arrays

array_intersect -- Computes the intersection of arrays

Description

array **array_intersect** (array array1, array array2 [, array ...])

array_intersect() returns an array containing all the values of array1 that are present in all the arguments. Note that keys are preserved.

Example 1. array_intersect() example

```
$array1 = array ("a" => "green", "red", "blue");
$array2 = array ("b" => "green", "yellow", "red");
$result = array_intersect ($array1, $array2);
```

This makes $result have

```
Array
(
    [a] => green
    [0] => red
)
```

> **Note:** Two elements are considered equal if and only if (string) $elem1 === (string) $elem2. In words: when the string representation is the same.

Warning
This was broken in PHP 4.0.4!

See also **array_intersect_assoc()**, **array_diff()**, **array_diff_assoc()**.

array_key_exists

(PHP 4 >= 4.1.0)

array_key_exists -- Checks if the given key or index exists in the array

Description

bool **array_key_exists** (mixed key, array search)

array_key_exists() returns **TRUE** if the given key is set in the array. key can be any value possible for an array index.

Example 1. array_key_exists() example

```
$search_array = array("first" => 1, "second" => 4);
if (array_key_exists("first", $search_array)) {
    echo "The 'first' element is in the array";
}
```

> **Note:** The name of this function is **key_exists()** in PHP version 4.0.6.

See also **isset()**.

array_keys

(PHP 4)

array_keys -- Return all the keys of an array

Description

array **array_keys** (array input [, mixed search_value])

array_keys() returns the keys, numeric and string, from the input array.

If the optional search_value is specified, then only the keys for that value are returned. Otherwise, all the keys from the input are returned.

Example 1. array_keys() example

```
$array = array (0 => 100, "color" => "red");
print_r(array_keys ($array));

$array = array ("blue", "red", "green", "blue", "blue");
print_r(array_keys ($array, "blue"));

$array = array ("color" => array("blue", "red", "green"), "size" => array("small", "medium", "large"));
print_r(array_keys ($array));
```

The printout of the program above will be:

```
Array
(
    [0] => 0
    [1] => color
)
```

```
    [1] => color
)
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
Array
(
    [0] => color
    [1] => size
)
```

Note: This function was added to PHP 4, below is an implementation for those still using PHP 3.

Example 2. Implementation of array_keys() for PHP 3 users

```
function array_keys ($arr, $term="") {
    $t = array();
    while (list($k,$v) = each($arr)) {
        if ($term && $v != $term) {
            continue;
        }
        $t[] = $k;
    }
    return $t;
}
```

See also **array_values()**.

array_map

(PHP 4 >= 4.0.6)

array_map -- Applies the callback to the elements of the given arrays

Description

array **array_map** (callback function, array arr1 [, array arr2...])

array_map() returns an array containing all the elements of arr1 after applying the callback function to each one. The number of parameters that the callback function accepts should match the number of arrays passed to the **array_map()**

Example 1. array_map() example

```
<?php
function cube($n) {
    return $n*$n*$n;
}

$a = array(1, 2, 3, 4, 5);
$b = array_map("cube", $a);
print_r($b);
?>
```

This makes $b have:

```
Array
(
    [0] => 1
    [1] => 8
    [2] => 27
    [3] => 64
    [4] => 125
)
```

Example 2. array_map() - using more arrays

```
<?php
function show_Spanish($n, $m) {
    return "The number $n is called $m in Spanish";
}

function map_Spanish($n, $m) {
    return array ($n => $m);
}

$a = array(1, 2, 3, 4, 5);
$b = array("uno", "dos", "tres", "cuatro", "cinco");

$c = array_map("show_Spanish", $a, $b);
print_r($c);

$d = array_map("map_Spanish", $a , $b);
print_r($d);
?>
```

This results:

This results:

```
// printout of $c
Array
(
    [0] => The number 1 is called uno in Spanish
    [1] => The number 2 is called dos in Spanish
    [2] => The number 3 is called tres in Spanish
    [3] => The number 4 is called cuatro in Spanish
    [4] => The number 5 is called cinco in Spanish
)

// printout of $d
Array
(
    [0] => Array
        (
            [1] => uno
        )

    [1] => Array
        (
            [2] => dos
        )

    [2] => Array
        (
            [3] => tres
        )

    [3] => Array
        (
            [4] => cuatro
        )

    [4] => Array
        (
            [5] => cinco
        )

)
```

Usually when using two or more arrays, they should be of equal length because the callback function is applied in parallel to the corresponding elements. If the arrays are of unequal length, the shortest one will be extended with empty elements.

An interesting use of this function is to construct an array of arrays, which can be easily performed by using **NULL** as the name of the callback function

Example 3. Creating an array of arrays

```
<?php
$a = array(1, 2, 3, 4, 5);
$b = array("one", "two", "three", "four", "five");
$c = array("uno", "dos", "tres", "cuatro", "cinco");

$d = array_map(null, $a, $b, $c);
print_r($d);
?>
```

The printout of the program above will be:

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => one
            [2] => uno
        )

    [1] => Array
        (
            [0] => 2
            [1] => two
            [2] => dos
        )

    [2] => Array
        (
            [0] => 3
            [1] => three
            [2] => tres
        )

    [3] => Array
        (
            [0] => 4
            [1] => four
            [2] => cuatro
        )

    [4] => Array
        (
            [0] => 5
```

```
    (
        [0] => 5
        [1] => five
        [2] => cinco
    )

)
```

See also **array_filter()**, **array_reduce()**, and **array_walk()**.

array_merge_recursive

(PHP 4 >= 4.0.1)

array_merge_recursive -- Merge two or more arrays recursively

Description

array **array_merge_recursive** (array array1, array array2 [, array ...])

array_merge_recursive() merges the elements of two or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

If the input arrays have the same string keys, then the values for these keys are merged together into an array, and this is done recursively, so that if one of the values is an array itself, the function will merge it with a corresponding entry in another array too. If, however, the arrays have the same numeric key, the later value will not overwrite the original value, but will be appended.

Example 1. array_merge_recursive() example

```
$ar1 = array ("color" => array ("favorite" => "red"), 5);
$ar2 = array (10, "color" => array ("favorite" => "green", "blue"));
$result = array_merge_recursive ($ar1, $ar2);
```

The $result will be:

```
Array
(
    [color] => Array
        (
            [favorite] => Array
                (
                    [0] => red
                    [1] => green
                )

            [0] => blue
        )

    [0] => 5
    [1] => 10
)
```

See also **array_merge()**.

array_merge

(PHP 4)

array_merge -- Merge two or more arrays

Description

array **array_merge** (array array1, array array2 [, array ...])

array_merge() merges the elements of two or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

If the input arrays have the same string keys, then the later value for that key will overwrite the previous one. If, however, the arrays contain numeric keys, the later value will **not** overwrite the original value, but will be appended.

Example 1. array_merge() example

```
$array1 = array ("color" => "red", 2, 4);
$array2 = array ("a", "b", "color" => "green", "shape" => "trapezoid", 4);
$result = array_merge ($array1, $array2);
```

The $result will be:

```
Array
(
    [color] => green
    [0] => 2
    [1] => 4
    [2] => a
    [3] => b
    [shape] => trapezoid
    [4] => 4
)
```

Example 2. Simple array_merge() example

```
$array1 = array();
$array2 = array(1 => "data");
$result = array_merge($array1, $array2);
```

Don't forget that numeric keys will be renumbered!

```
Array
(
    [0] => data
)
```

If you want to completely preserve the arrays and just want to append them to each other, use the + operator:

```
$array1 = array();
$array2 = array(1 => "data");
$result = $array1 + $array2;
```

The numeric key will be preserved and thus the association remains.

```
Array
(
    [1] => data
)
```

> **Note:** Shared keys will be overwritten on a first-come first-served basis.

See also **array_merge_recursive()**.

array_multisort

(PHP 4)

array_multisort -- Sort multiple or multi-dimensional arrays

Description

bool **array_multisort** (array ar1 [, mixed arg [, mixed ... [, array ...]]])

array_multisort() can be used to sort several arrays at once or a multi-dimensional array according by one of more dimensions. It maintains key association when sorting.

The input arrays are treated as columns of a table to be sorted by rows - this resembles the functionality of SQL ORDER BY clause. The first array is the primary one to sort by. The rows (values) in that array that compare the same are sorted by the next input array, and so on.

The argument structure of this function is a bit unusual, but flexible. The very first argument has to be an array. Subsequently, each argument can be either an array or a sorting flag from the following lists.

Sorting order flags:

- SORT_ASC - sort in ascending order
- SORT_DESC - sort in descending order

Sorting type flags:

- SORT_REGULAR - compare items normally
- SORT_NUMERIC - compare items numerically
- SORT_STRING - compare items as strings

No two sorting flags of the same type can be specified after each array. The sorting flags specified after an array argument apply only to that array - they are reset to default SORT_ASC and SORT_REGULAR before each new array argument.

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

Example 1. Sorting multiple arrays

```
$ar1 = array ("10", 100, 100, "a");
$ar2 = array (1, 3, "2", 1);
array_multisort ($ar1, $ar2);
```

In this example, after sorting, the first array will contain 10, "a", 100, 100. The second array will contain 1, 1, "2", 3. The entries in the second array corresponding to the identical entries in the first array (100 and 100) were sorted as well.

Example 2. Sorting multi-dimensional array

```
$ar = array (array ("10", 100, 100, "a"), array (1, 3, "2", 1));
array_multisort ($ar[0], SORT_ASC, SORT_STRING,
                 $ar[1], SORT_NUMERIC, SORT_DESC);
```

In this example, after sorting, the first array will contain 10, 100, 100, "a" (it was sorted as strings in ascending order), and the second one will contain 1, 3, "2", 1 (sorted as numbers, in descending order).

array_pad

(PHP 4)

array_pad -- Pad array to the specified length with a value

array_pad -- Pad array to the specified length with a value

Description

array **array_pad** (array input, int pad_size, mixed pad_value)

array_pad() returns a copy of the input padded to size specified by pad_size with value pad_value. If pad_size is positive then the array is padded on the right, if it's negative then on the left. If the absolute value of pad_size is less than or equal to the length of the input then no padding takes place.

Example 1. array_pad() example

```
$input = array (12, 10, 9);

$result = array_pad ($input, 5, 0);
// result is array (12, 10, 9, 0, 0)

$result = array_pad ($input, -7, -1);
// result is array (-1, -1, -1, -1, 12, 10, 9)

$result = array_pad ($input, 2, "noop");
// not padded
```

array_pop

(PHP 4)

array_pop -- Pop the element off the end of array

Description

mixed **array_pop** (array array)

array_pop() pops and returns the last value of the array, shortening the array by one element. If array is empty (or is not an array), **NULL** will be returned.

Example 1. array_pop() example

```
$stack = array ("orange", "banana", "apple", "raspberry");
$fruit = array_pop ($stack);
```

After this, $stack will have only 3 elements:

```
Array
(
    [0] => orange
    [1] => banana
    [2] => apple
)
```

and rasberry will be assigned to $fruit.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as 0 or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

See also **array_push()**, **array_shift()**, and **array_unshift()**.

array_push

(PHP 4)

array_push -- Push one or more elements onto the end of array

Description

int **array_push** (array array, mixed var [, mixed ...])

array_push() treats array as a stack, and pushes the passed variables onto the end of array. The length of array increases by the number of variables pushed. Has the same effect as:

```
$array[] = $var;
```

repeated for each var.

Returns the new number of elements in the array.

Example 1. array_push() example

```
$stack = array ("orange", "banana");
array_push ($stack, "apple", "raspberry");
```

This example would result in $stack having the following elements:

```
Array
(
    [0] => orange
    [1] => banana
```

```
    [0] => orange
    [1] => banana
    [2] => apple
    [3] => raspberry
)
```

See also **array_pop()**, **array_shift()**, and **array_unshift()**.

array_rand

(PHP 4)

array_rand -- Pick one or more random entries out of an array

Description

mixed **array_rand** (array input [, int num_req])

array_rand() is rather useful when you want to pick one or more random entries out of an array. It takes an input array and an optional argument num_req which specifies how many entries you want to pick - if not specified, it defaults to 1.

If you are picking only one entry, **array_rand()** returns the key for a random entry. Otherwise, it returns an array of keys for the random entries. This is done so that you can pick random keys as well as values out of the array.

Don't forget to call **srand()** to seed the random number generator.

Example 1. array_rand() example

```
srand ((float) microtime() * 10000000);
$input = array ("Neo", "Morpheus", "Trinity", "Cypher", "Tank");
$rand_keys = array_rand ($input, 2);
print $input[$rand_keys[0]]."\n";
print $input[$rand_keys[1]]."\n";
```

array_reduce

(PHP 4 >= 4.0.5)

array_reduce -- Iteratively reduce the array to a single value using a callback function

Description

mixed **array_reduce** (array input, callback function [, int initial])

array_reduce() applies iteratively the function function to the elements of the array input, so as to reduce the array to a single value. If the optional initial is available, it will be used at the beginning of the process, or as a final result in case the array is empty.

Example 1. array_reduce() example

```
function rsum($v, $w) {
    $v += $w;
    return $v;
}

function rmul($v, $w) {
    $v *= $w;
    return $v;
}

$a = array(1, 2, 3, 4, 5);
$x = array();
$b = array_reduce($a, "rsum");
$c = array_reduce($a, "rmul", 10);
$d = array_reduce($x, "rsum", 1);
```

This will result in $b containing 15, $c containing 1200 (= 1*2*3*4*5*10), and $d containing 1.

See also **array_filter()** and **array_map()**.

array_reverse

(PHP 4)

array_reverse -- Return an array with elements in reverse order

Description

array **array_reverse** (array array [, bool preserve_keys])

array_reverse() takes input array and returns a new array with the order of the elements reversed, preserving the keys if preserve_keys is **TRUE**.

Example 1. array_reverse() example

```
$input  = array ("php", 4.0, array ("green", "red"));
$result = array_reverse ($input);
$result_keyed = array_reverse ($input, TRUE);
```

```
$result = array_reverse ($input);
$result_keyed = array_reverse ($input, TRUE);
```

This makes both $result and $result_keyed have the same elements, but note the difference between the keys. The printout of $result and $result_keyed will be:

```
Array
(
    [0] => Array
        (
            [0] => green
            [1] => red
        )

    [1] => 4
    [2] => php
)
Array
(
    [2] => Array
        (
            [0] => green
            [1] => red
        )

    [1] => 4
    [0] => php
)
```

Note: The second parameter was added in PHP 4.0.3.

array_search

(PHP 4 >= 4.0.5)

array_search -- Searches the array for a given value and returns the corresponding key if successful

Description

mixed **array_search** (mixed needle, array haystack [, bool strict])

Searches haystack for needle and returns the key if it is found in the array, **FALSE** otherwise.

Note: Prior to PHP 4.2.0, **array_search()** returns **NULL** on failure instead of **FALSE**.

If the optional third parameter strict is set to **TRUE** then the **array_search()** will also check the types of the needle in the haystack.

Warning
This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as 0 or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

See also **array_keys()** and **in_array()**.

array_shift

(PHP 4)

array_shift -- Shift an element off the beginning of array

Description

mixed **array_shift** (array array)

array_shift() shifts the first value of the array off and returns it, shortening the array by one element and moving everything down. All numerical array keys will be modified to start counting from zero while literal keys won't be touched. If array is empty (or is not an array), **NULL** will be returned.

Example 1. array_shift() example

```
$stack = array ("orange", "banana", "apple", "raspberry");
$fruit = array_shift ($stack);
```

This would result in $stack having 3 elements left:

```
Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
)
```

and orange will be assigned to $fruit.

See also **array_unshift()**, **array_push()**, and **array_pop()**.

array_slice

(PHP 4)

(PHP 4)

array_slice -- Extract a slice of the array

Description

array **array_slice** (array array, int offset [, int length])

array_slice() returns the sequence of elements from the array array as specified by the offset and length parameters.

If offset is positive, the sequence will start at that offset in the array. If offset is negative, the sequence will start that far from the end of the array.

If length is given and is positive, then the sequence will have that many elements in it. If length is given and is negative then the sequence will stop that many elements from the end of the array. If it is omitted, then the sequence will have everything from offset up until the end of the array.

Note that **array_slice()** will ignore array keys, and will calculate offsets and lengths based on the actual positions of elements within the array.

Example 1. array_slice() examples

```
$input = array ("a", "b", "c", "d", "e");

$output = array_slice ($input, 2);      // returns "c", "d", and "e"
$output = array_slice ($input, 2, -1);  // returns "c", "d"
$output = array_slice ($input, -2, 1);  // returns "d"
$output = array_slice ($input, 0, 3);   // returns "a", "b", and "c"
```

See also **array_splice()**.

array_splice

(PHP 4)

array_splice -- Remove a portion of the array and replace it with something else

Description

array **array_splice** (array input, int offset [, int length [, array replacement]])

array_splice() removes the elements designated by offset and length from the input array, and replaces them with the elements of the replacement array, if supplied. It returns an array containing the extracted elements.

If offset is positive then the start of removed portion is at that offset from the beginning of the input array. If offset is negative then it starts that far from the end of the input array.

If length is omitted, removes everything from offset to the end of the array. If length is specified and is positive, then that many elements will be removed. If length is specified and is negative then the end of the removed portion will be that many elements from the end of the array. Tip: to remove everything from offset to the end of the array when replacement is also specified, use count($input) for length.

If replacement array is specified, then the removed elements are replaced with elements from this array. If offset and length are such that nothing is removed, then the elements from the replacement array are inserted in the place specified by the offset. Tip: if the replacement is just one element it is not necessary to put array() around it, unless the element is an array itself.

The following equivalences hold:

```
array_push ($input, $x, $y)     array_splice ($input, count ($input), 0,
                                             array ($x, $y))
array_pop ($input)              array_splice ($input, -1)
array_shift ($input)            array_splice ($input, 0, 1)
array_unshift ($input, $x, $y)  array_splice ($input, 0, 0, array ($x, $y))
$input[$x] = $y                 array_splice ($input, $x, 1, $y)
```

Returns the array consisting of removed elements.

Example 1. array_splice() examples

```
$input = array ("red", "green", "blue", "yellow");
array_splice ($input, 2);
// $input is now array ("red", "green")

$input = array ("red", "green", "blue", "yellow");
array_splice ($input, 1, -1);
// $input is now array ("red", "yellow")

$input = array ("red", "green", "blue", "yellow");
array_splice ($input, 1, count($input), "orange");
// $input is now array ("red", "orange")

$input = array ("red", "green", "blue", "yellow");
array_splice ($input, -1, 1, array("black", "maroon"));
// $input is now array ("red", "green",
//         "blue", "black", "maroon")
```

See also **array_slice()**.

array_sum

(PHP 4 >= 4.0.4)

(PHP 4 >= 4.0.4)

array_sum -- Calculate the sum of values in an array.

Description

mixed **array_sum** (array array)

array_sum() returns the sum of values in an array as an integer or float.

Example 1. array_sum() examples

```
$a = array(2, 4, 6, 8);
echo "sum(a) = ".array_sum($a)."\n";

$b = array("a"=>1.2,"b"=>2.3,"c"=>3.4);
echo "sum(b) = ".array_sum($b)."\n";
```

The printout of the program above will be:

```
sum(a) = 20
sum(b) = 6.9
```

> **Note:** PHP versions prior to 4.0.6 modified the passed array itself and converted strings to numbers (which most of the time converted them to zero, depending on their value).

array_unique

(PHP 4 >= 4.0.1)

array_unique -- Removes duplicate values from an array

Description

array **array_unique** (array array)

array_unique() takes input array and returns a new array without duplicate values.

Note that keys are preserved. **array_unique()** sorts the values treated as string at first, then will keep the first key encountered for every value, and ignore all following keys. It does not mean that the key of the first related value from the unsorted array will be kept.

> **Note:** Two elements are considered equal if and only if (string) $elem1 === (string) $elem2. In words: when the string representation is the same.
>
> The first element will be used.

Warning
This was broken in PHP 4.0.4!

Example 1. array_unique() example

```
$input = array ("a" => "green", "red", "b" => "green", "blue", "red");
$result = array_unique ($input);
print_r($result);
```

This will output:

```
Array
(
    [b] => green
    [1] => blue
    [2] => red
)
```

Example 2. array_unique() and types

```
$input = array (4,"4","3",4,3,"3");
$result = array_unique ($input);
var_dump($result);
```

The printout of the program above will be (PHP 4.0.6):

```
array(2) {
  [3]=>
  int(4)
  [4]=>
  int(3)
}
```

array_unshift

(PHP 4)

array_unshift -- Prepend one or more elements to the beginning of array

array_unshift -- Prepend one or more elements to the beginning of array

Description

int **array_unshift** (array array, mixed var [, mixed ...])

array_unshift() prepends passed elements to the front of the array. Note that the list of elements is prepended as a whole, so that the prepended elements stay in the same order. All numerical array keys will be modified to start counting from zero while literal keys won't be touched.

Returns the new number of elements in the array.

Example 1. array_unshift() example

```
$queue = array ("orange", "banana");
array_unshift ($queue, "apple", "raspberry");
```

This would result in $queue having the following elements:

```
Array
(
    [0] => apple
    [1] => raspberry
    [2] => orange
    [3] => banana
)
```

See also **array_shift()**, **array_push()**, and **array_pop()**.

array_values

(PHP 4)

array_values -- Return all the values of an array

Description

array **array_values** (array input)

array_values() returns all the values from the input array.

Example 1. array_values() example

```
$array = array ("size" => "XL", "color" => "gold");
print_r(array_values ($array));
```

This will output:

```
Array
(
    [0] => XL
    [1] => gold
)
```

Note: This function was added to PHP 4, below is an implementation for those still using PHP 3.

Example 2. Implementation of array_values() for PHP 3 users

```
function array_values ($arr) {
    $t = array();
    while (list($k, $v) = each ($arr)) {
        $t[] = $v;
    }
    return $t;
}
```

See also **array_keys()**.

array_walk

(PHP 3>= 3.0.3, PHP 4)

array_walk -- Apply a user function to every member of an array

Description

int **array_walk** (array array, callback function [, mixed userdata])

Applies the user-defined function function to each element of the array array. Typically, function takes on two parameters. The array parameter's value being the first, and the key/index second. If the optional userdata parameter is supplied, it will be passed as the third parameter to the callback function.

If function function requires more parameters than given to it, an error of level E_WARNING will be generated each time **array_walk()** calls function. These warnings may be suppressed by prepending the PHP error operator @ to the **array_walk()** call, or by using **error_reporting()**.

Note: If function needs to be working with the actual values of the array, specify the first parameter of function as a reference. Then, any changes made to those elements will be made in the original array itself.

Note: If function needs to be working with the actual values of the array, specify the first parameter of function as a reference. Then, any changes made to those elements will be made in the original array itself.

Note: Passing the key and userdata to function was added in 4.0.0

array_walk() is not affected by the internal array pointer of array. **array_walk()** will walk through the entire array regardless of pointer position. To reset the pointer, use **reset()**. In PHP 3, **array_walk()** resets the pointer.

Users may not change the array itself from the callback function. e.g. Add/delete elements, unset elements, etc. If the array that **array_walk()** is applied to is changed, the behavior of this function is undefined, and unpredictable.

Example 1. array_walk() example

```
<?php
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");

function test_alter (&$item1, $key, $prefix) {
    $item1 = "$prefix: $item1";
}

function test_print ($item2, $key) {
    echo "$key. $item2<br>\n";
}

echo "Before ...:\n";
array_walk ($fruits, 'test_print');

array_walk ($fruits, 'test_alter', 'fruit');
echo "... and after:\n";

array_walk ($fruits, 'test_print');
?>
```

The printout of the program above will be:

```
Before ...:
d. lemon
a. orange
b. banana
c. apple
... and after:
d. fruit: lemon
a. fruit: orange
b. fruit: banana
c. fruit: apple
```

See also **list()**, foreach, **each()**, and **call_user_func_array()**.

array

(PHP 3, PHP 4)

array -- Create an array

Description

array **array** ([mixed ...])

Returns an array of the parameters. The parameters can be given an index with the => operator.

Note: **array()** is a language construct used to represent literal arrays, and not a regular function.

Syntax "index => values", separated by commas, define index and values. index may be of type string or numeric. When index is omitted, a integer index is automatically generated, starting at 0. If index is an integer, next generated index will be the biggest integer index + 1. Note that when two identical index are defined, the last overwrite the first.

The following example demonstrates how to create a two-dimensional array, how to specify keys for associative arrays, and how to skip-and-continue numeric indices in normal arrays.

Example 1. array() example

```
$fruits = array (
    "fruits"  => array ("a"=>"orange", "b"=>"banana", "c"=>"apple"),
    "numbers" => array (1, 2, 3, 4, 5, 6),
    "holes"   => array ("first", 5 => "second", "third")
);
```

Example 2. Automatic index with array()

```
$array = array( 1, 1, 1, 1,  1, 8=>1,  4=>1, 19, 3=>13);
print_r($array);
```

will display :

```
Array
(
    [0] => 1
    [1] => 1
    [2] => 1
    [3] => 13
```

```
    [2] => 1
    [3] => 13
    [4] => 1
    [8] => 1
    [9] => 19
)
```

Note that index '3' is defined twice, and keep its final value of 13. Index 4 is defined after index 8, and next generated index (value 19) is 9, since biggest index was 8.

This example creates a 1-based array.

Example 3. 1-based index with array()

```
$firstquarter  = array(1 => 'January', 'February', 'March');
print_r($firstquarter);
```

will display :

```
Array
(
    [1] => 'January'
    [2] => 'February'
    [3] => 'March'
)
```

See also **array_pad()**, **list()**, and **range()**.

arsort

(PHP 3, PHP 4)

arsort -- Sort an array in reverse order and maintain index association

Description

void **arsort** (array array [, int sort_flags])

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Example 1. arsort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
arsort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\n";
}
```

This example would display:

```
a = orange
d = lemon
b = banana
c = apple
```

The fruits have been sorted in reverse alphabetical order, and the index associated with each element has been maintained.

You may modify the behavior of the sort using the optional parameter sort_flags, for details see **sort()**.

See also **asort()**, **rsort()**, **ksort()**, and **sort()**.

asort

(PHP 3, PHP 4)

asort -- Sort an array and maintain index association

Description

void **asort** (array array [, int sort_flags])

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Example 1. asort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
asort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\n";
}
```

This example would display:

```
c = apple
b = banana
d = lemon
a = orange
```

```
a = orange
```

The fruits have been sorted in alphabetical order, and the index associated with each element has been maintained.

You may modify the behavior of the sort using the optional parameter sort_flags, for details see **sort()**.

See also **arsort()**, **rsort()**, **ksort()**, and **sort()**.

compact

(PHP 4)

compact -- Create array containing variables and their values

Description

array **compact** (mixed varname [, mixed ...])

compact() takes a variable number of parameters. Each parameter can be either a string containing the name of the variable, or an array of variable names. The array can contain other arrays of variable names inside it; **compact()** handles it recursively.

For each of these, **compact()** looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key. In short, it does the opposite of **extract()**. It returns the output array with all the variables added to it.

Any strings that are not set will simply be skipped.

Example 1. compact() example

```
$city = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";

$location_vars = array ("city", "state");

$result = compact ("event", "nothing_here", $location_vars);
```

After this, $result will be:

```
Array
(
    [event] => SIGGRAPH
    [city] => San Francisco
    [state] => CA
)
```

See also **extract()**.

count

(PHP 3, PHP 4)

count -- Count elements in a variable

Description

int **count** (mixed var)

Returns the number of elements in var, which is typically an **array** (since anything else will have one element).

If var is not an array, 1 will be returned (exception: count(**NULL**) equals 0).

Warning
count() may return 0 for a variable that isn't set, but it may also return 0 for a variable that has been initialized with an empty array. Use **isset()** to test if a variable is set.

Please see the Arrays section of the manual for a detailed explanation of how arrays are implemented and used in PHP.

Example 1. count() example

```
$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count ($a);
// $result == 3

$b[0] = 7;
$b[5] = 9;
$b[10] = 11;
$result = count ($b);
// $result == 3;
```

Note: The **sizeof()** function is an alias for **count()**.

See also **is_array()**, **isset()**, and **strlen()**.

current

(PHP 3, PHP 4)

(PHP 3, PHP 4)

current -- Return the current element in an array

Description

mixed **current** (array array)

Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.

The **current()** function simply returns the array element that's currently being pointed by the internal pointer. It does not move the pointer in any way. If the internal pointer points beyond the end of the elements list, **current()** returns **FALSE**.

Warning
If the array contains empty elements (0 or "", the empty string) then this function will return **FALSE** for these elements as well. This makes it impossible to determine if you are really at the end of the list in such an array using **current()**. To properly traverse an array that may contain empty elements, use the **each()** function.

See also **end()**, **next()**, **prev()**, and **reset()**.

each

(PHP 3, PHP 4)

each -- Return the current key and value pair from an array and advance the array cursor

Description

array **each** (array array)

Returns the current key and value pair from the array array and advances the array cursor. This pair is returned in a four-element array, with the keys 0, 1, key, and value. Elements 0 and key contain the key name of the array element, and 1 and value contain the data.

If the internal pointer for the array points past the end of the array contents, **each()** returns **FALSE**.

Example 1. each() examples

```
$foo = array ("bob", "fred", "jussi", "jouni", "egon", "marliese");
$bar = each ($foo);
```

$bar now contains the following key/value pairs:

- 0 => 0
- 1 => 'bob'
- key => 0
- value => 'bob'

```
$foo = array ("Robert" => "Bob", "Seppo" => "Sepi");
$bar = each ($foo);
```

$bar now contains the following key/value pairs:

- 0 => 'Robert'
- 1 => 'Bob'
- key => 'Robert'
- value => 'Bob'

each() is typically used in conjunction with **list()** to traverse an array; for instance, $_POST:

Example 2. Traversing $_POST with each()

```
echo "Values submitted via POST method:<br />\n";
reset ($_POST);
while (list ($key, $val) = each ($_POST)) {
    echo "$key => $val<br />\n";
}
```

After **each()** has executed, the array cursor will be left on the next element of the array, or on the last element if it hits the end of the array. You have to use **reset()** if you want to traverse the array again using each.

See also **key()**, **list()**, **current()**, **reset()**, **next()**, **prev()**, and foreach.

end

(PHP 3, PHP 4)

end -- Set the internal pointer of an array to its last element

Description

mixed **end** (array array)

end() advances array's internal pointer to the last element, and returns that element.

Example 1. A simple end() example

Example 1. A simple end() example

```
<?php

  $fruits = array('apple','banana','cranberry');

  print end($fruits); // cranberry

?>
```

See also **current()**, **each()**, **next()**, and **reset()**.

extract

(PHP 3>= 3.0.7, PHP 4)

extract -- Import variables into the current symbol table from an array

Description

int **extract** (array var_array [, int extract_type [, string prefix]])

This function is used to import variables from an array into the current symbol table. It takes an associative array var_array and treats keys as variable names and values as variable values. For each key/value pair it will create a variable in the current symbol table, subject to extract_type and prefix parameters.

> **Note:** Beginning with version 4.0.5, this function returns the number of variables extracted.
>
> **Note:** EXTR_IF_EXISTS and EXTR_PREFIX_IF_EXISTS was introduced in version 4.2.0.
>
> **Note:** EXTR_REFS was introduced in version 4.3.0.

extract() checks each key to see whether it has a valid variable name. It also checks for collisions with existing variables in the symbol table. The way invalid/numeric keys and collisions are treated is determined by the extract_type. It can be one of the following values:

EXTR_OVERWRITE

> If there is a collision, overwrite the existing variable.

EXTR_SKIP

> If there is a collision, don't overwrite the existing variable.

EXTR_PREFIX_SAME

> If there is a collision, prefix the variable name with prefix.

EXTR_PREFIX_ALL

> Prefix all variable names with prefix. Beginning with PHP 4.0.5, this includes numeric variables as well.

EXTR_PREFIX_INVALID

> Only prefix invalid/numeric variable names with prefix. This flag was added in PHP 4.0.5.

EXTR_IF_EXISTS

> Only overwrite the variable if it already exists in the current symbol table, otherwise do nothing. This is useful for defining a list of valid variables and then extracting only those variables you have defined out of $_REQUEST, for example. This flag was added in PHP 4.2.0.

EXTR_PREFIX_IF_EXISTS

> Only create prefixed variable names if the non-prefixed version of the same variable exists in the current symbol table. This flag was added in PHP 4.2.0.

EXTR_REFS

> Extracts variables as references. This effectively means that the values of the imported variables are still referencing the values of the var_array parameter. You can use this flag on its own or combine it with any other flag by OR'ing the extract_type. This flag was added in PHP 4.3.0.

If extract_type is not specified, it is assumed to be EXTR_OVERWRITE.

Note that prefix is only required if extract_type is EXTR_PREFIX_SAME, EXTR_PREFIX_ALL, EXTR_PREFIX_INVALID or EXTR_PREFIX_IF_EXISTS. If the prefixed result is not a valid variable name, it is not imported into the symbol table.

extract() returns the number of variables successfully imported into the symbol table.

A possible use for **extract()** is to import into the symbol table variables contained in an associative array returned by **wddx_deserialize()**.

Example 1. extract() example

```
<?php

/* Suppose that $var_array is an array returned from
   wddx_deserialize */

$size = "large";
$var_array = array ("color" => "blue",
                    "size"  => "medium",
                    "shape" => "sphere");
extract ($var_array, EXTR_PREFIX_SAME, "wddx");

print "$color, $size, $shape, $wddx_size\n";

?>
```

The above example will produce:

```
blue, large, sphere, medium
```

The above example will produce:

```
blue, large, sphere, medium
```

The $size wasn't overwritten, because we specified EXTR_PREFIX_SAME, which resulted in $wddx_size being created. If EXTR_SKIP was specified, then $wddx_size wouldn't even have been created. EXTR_OVERWRITE would have caused $size to have value "medium", and EXTR_PREFIX_ALL would result in new variables being named $wddx_color, $wddx_size, and $wddx_shape.

You must use an associative array, a numerically indexed array will not produce results unless you use EXTR_PREFIX_ALL or EXTR_PREFIX_INVALID.

See also **compact()**.

in_array

(PHP 4)

in_array -- Return **TRUE** if a value exists in an array

Description

bool **in_array** (mixed needle, array haystack [, bool strict])

Searches haystack for needle and returns **TRUE** if it is found in the array, **FALSE** otherwise.

If the third parameter strict is set to **TRUE** then the **in_array()** function will also check the types of the needle in the haystack.

> **Note:** If needle is a string, the comparison is done in a case-sensitive manner.
>
> **Note:** In PHP versions before 4.2.0 needle was not allowed to be an array.

Example 1. in_array() example

```
<?php
$os = array ("Mac", "NT", "Irix", "Linux");
if (in_array ("Irix", $os)) {
    print "Got Irix";
}
if (in_array ("mac", $os)) {
    print "Got mac";
}
?>
```

The second condition fails because **in_array()** is case-sensitive, so the program above will display:

```
Got Irix
```

Example 2. in_array() with strict example

```
<?php
$a = array('1.10', 12.4, 1.13);

if (in_array('12.4', $a, TRUE))
    echo "'12.4' found with strict check\n";
if (in_array(1.13, $a, TRUE))
    echo "1.13 found with strict check\n";
?>
```

This will display:

```
1.13 found with strict check
```

Example 3. in_array() with an array as needle

```
<?php
$a = array(array('p', 'h'), array('p', 'r'), 'o');

if (in_array(array ('p', 'h'), $a))
    echo "'ph' was found\n";
if (in_array(array ('f', 'i'), $a))
    echo "'fi' was found\n";
if (in_array('o', $a))
    echo "'o' was found\n";
?>

// This will output:

'ph' was found
'o' was found
```

See also **array_search()**.

key

(PHP 3, PHP 4)

key -- Fetch a key from an associative array

Description

mixed **key** (array array)

Description

mixed **key** (array array)

key() returns the index element of the current array position.

See also **current()** and **next()**.

krsort

(PHP 3>= 3.0.13, PHP 4)

krsort -- Sort an array by key in reverse order

Description

int **krsort** (array array [, int sort_flags])

Sorts an array by key in reverse order, maintaining key to data correlations. This is useful mainly for associative arrays.

Example 1. krsort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
krsort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\n";
}
```

This example would display:

```
d = lemon
c = apple
b = banana
a = orange
```

You may modify the behavior of the sort using the optional parameter sort_flags, for details see **sort()**.

See also **asort()**, **arsort()**, **ksort()**, **sort()**, **natsort()**, and **rsort()**.

ksort

(PHP 3, PHP 4)

ksort -- Sort an array by key

Description

int **ksort** (array array [, int sort_flags])

Sorts an array by key, maintaining key to data correlations. This is useful mainly for associative arrays.

Example 1. ksort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
ksort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\n";
}
```

This example would display:

```
a = orange
b = banana
c = apple
d = lemon
```

You may modify the behavior of the sort using the optional parameter sort_flags, for details see **sort()**.

See also **asort()**, **arsort()**, **krsort()**, **uksort()**, **sort()**, **natsort()**, and **rsort()**.

> **Note:** The second parameter was added in PHP 4.

list

(PHP 3, PHP 4)

list -- Assign variables as if they were an array

Description

void **list** (mixed ...)

Like **array()**, this is not really a function, but a language construct. **list()** is used to assign a list of variables in one operation.

> **Note:** **list()** only works on numerical arrays and assumes the numerical indices start at 0.

Note: **list()** only works on numerical arrays and assumes the numerical indices start at 0.

Example 1. list() examples

```
<?php

$info = array('coffee', 'brown', 'caffeine');

// Listing all the variables
list($drink, $color, $power) = $info;
print "$drink is $color and $power makes it special.\n";

// Listing some of them
list($drink, , $power) = $info;
print "$drink has $power.\n";

// Or let's skip to only the third one
list( , , $power) = $info;
print "I need $power!\n";

?>
```

Example 2. An example use of list()

```
<table>
 <tr>
  <th>Employee name</th>
  <th>Salary</th>
 </tr>

<?php

$result = mysql_query ("SELECT id, name, salary FROM employees",$conn);
while (list ($id, $name, $salary) = mysql_fetch_row ($result)) {
    print (" <tr>\n".
           "  <td><a href=\"info.php?id=$id\">$name</a></td>\n".
           "  <td>$salary</td>\n".
           " </tr>\n");
}

?>

</table>
```

Warning
list() assigns the values starting with the right-most parameter. If you are using plain variables, you don't have to worry about this. But if you are using arrays with indices you usually expect the order of the indices in the array the same you wrote in the **list()** from left to right; which it isn't. It's assigned in the reverse order.

Example 3. Using list() with array indices

```
<?php

$info = array('coffee', 'brown', 'caffeine');

list($a[0], $a[1], $a[2]) = $info;

var_dump($a);

?>
```

Gives the following output (note the order of the elements compared in which order they were written in the **list()** syntax):

```
array(3) {
  [2]=>
  string(8) "caffeine"
  [1]=>
  string(5) "brown"
  [0]=>
  string(6) "coffee"
}
```

See also **each()**, **array()** and **extract()**.

natcasesort

(PHP 4)

natcasesort -- Sort an array using a case insensitive "natural order" algorithm

Description

void **natcasesort** (array array)

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would. This is described as a "natural ordering".

natcasesort() is a case insensitive version of **natsort()**. See **natsort()** for an example of the difference between this algorithm and the regular computer string sorting algorithms.

natcasesort() is a case insensitive version of **natsort()**. See **natsort()** for an example of the difference between this algorithm and the regular computer string sorting algorithms.

For more information see: Martin Pool's Natural Order String Comparison page.

See also **sort()**, **natsort()**, **strnatcmp()**, and **strnatcasecmp()**.

natsort

(PHP 4)

natsort -- Sort an array using a "natural order" algorithm

Description

void **natsort** (array array)

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would. This is described as a "natural ordering". An example of the difference between this algorithm and the regular computer string sorting algorithms (used in **sort()**) can be seen below:

Example 1. natsort() example

```
<?php
$array1 = $array2 = array ("img12.png", "img10.png", "img2.png", "img1.png");

sort($array1);
echo "Standard sorting\n";
print_r($array1);

natsort($array2);
echo "\nNatural order sorting\n";
print_r($array2);
?>
```

The code above will generate the following output:

```
Standard sorting
Array
(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
)

Natural order sorting
Array
(
    [3] => img1.png
    [2] => img2.png
    [1] => img10.png
    [0] => img12.png
)
```

For more information see: Martin Pool's Natural Order String Comparison page.

> **Note:** If you're wanting to maintain index/value associations, consider using usort($arr, 'strnatcmp').

See also **natcasesort()**, **strnatcmp()**, and **strnatcasecmp()**.

next

(PHP 3, PHP 4)

next -- Advance the internal array pointer of an array

Description

mixed **next** (array array)

Returns the array element in the next place that's pointed by the internal array pointer, or **FALSE** if there are no more elements.

next() behaves like **current()**, with one difference. It advances the internal array pointer one place forward before returning the element. That means it returns the next array element and advances the internal array pointer by one. If advancing the internal array pointer results in going beyond the end of the element list, **next()** returns **FALSE**.

Warning
If the array contains empty elements, or elements that have a key value of 0 then this function will return **FALSE** for these elements as well. To properly traverse an array which may contain empty elements or elements with key values of 0 see the **each()** function.

See also **current()**, **end()**, **prev()**, and **reset()**.

pos

(PHP 3, PHP 4)

pos -- Get the current element from an array

Description

pos -- Get the current element from an array

Description

mixed **pos** (array array)

This is an alias for **current()**.

See also **end()**, **next()**, **prev()**, and **reset()**.

prev

(PHP 3, PHP 4)

prev -- Rewind the internal array pointer

Description

mixed **prev** (array array)

Returns the array element in the previous place that's pointed by the internal array pointer, or **FALSE** if there are no more elements.

Warning
If the array contains empty elements then this function will return **FALSE** for these elements as well. To properly traverse an array which may contain empty elements see the **each()** function.

prev() behaves just like **next()**, except it rewinds the internal array pointer one place instead of advancing it.

See also **current()**, **end()**, **next()**, and **reset()**.

range

(PHP 3>= 3.0.8, PHP 4)

range -- Create an array containing a range of elements

Description

array **range** (mixed low, mixed high [, int step])

range() returns an array of elements from low to high, inclusive. If low > high, the sequence will be from high to low.

> **New parameter:** The optional step parameter was added in 5.0.0.

If a step value is given, it will be used as the increment between elements in the sequence. step should be given as a positive number. If not specified, step will default to 1.

Example 1. range() examples

```
<?php
// array(0,1,2,3,4,5,6,7,8,9)
foreach(range(0, 9) as $number) {
    echo $number;
}

// The step parameter was introduced in 5.0.0
// array(0,10,20,30,40,50,60,70,80,90,100)
foreach(range(0, 100, 10) as $number) {
    echo $number;
}

// Use of characters introduced in 4.1.0
// array('a','b','c','d','e','f','g','h','i');
foreach(range('a', 'i') as $letter) {
    echo $letter;
}
// array('c','b','a');
foreach(range('c', 'a') as $letter) {
    echo $letter;
}
?>
```

> **Note:** Prior to version 4.1.0 the **range()** function only generated incrementing integer arrays. Support for character sequences and decrementing arrays was added in 4.1.0.

See also **shuffle()** and foreach.

reset

(PHP 3, PHP 4)

reset -- Set the internal pointer of an array to its first element

Description

mixed **reset** (array array)

reset() rewinds array's internal pointer to the first element.

reset() rewinds array's internal pointer to the first element.

reset() returns the value of the first array element.

See also **current()**, **each()**, **next()**, and **prev()**.

rsort

(PHP 3, PHP 4)

rsort -- Sort an array in reverse order

Description

void **rsort** (array array [, int sort_flags])

This function sorts an array in reverse order (highest to lowest).

Example 1. rsort() example

```
$fruits = array ("lemon", "orange", "banana", "apple");
rsort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "$key = $val\n";
}
```

This example would display:

```
0 = orange
1 = lemon
2 = banana
3 = apple
```

The fruits have been sorted in reverse alphabetical order.

You may modify the behavior of the sort using the optional parameter sort_flags, for details see **sort()**.

See also **arsort()**, **asort()**, **ksort()**, **sort()**, and **usort()**.

shuffle

(PHP 3>= 3.0.8, PHP 4)

shuffle -- Shuffle an array

Description

void **shuffle** (array array)

This function shuffles (randomizes the order of the elements in) an array. You must use **srand()** to seed this function.

Example 1. shuffle() example

```
$numbers = range (1,20);
srand ((float)microtime()*1000000);
shuffle ($numbers);
while (list (, $number) = each ($numbers)) {
    echo "$number ";
}
```

See also **arsort()**, **asort()**, **ksort()**, **rsort()**, **sort()**, and **usort()**.

sizeof

(PHP 3, PHP 4)

sizeof -- Get the number of elements in variable

Description

int **sizeof** (mixed var)

The **sizeof()** function is an alias for **count()**.

See also **count()**.

sort

(PHP 3, PHP 4)

sort -- Sort an array

Description

void **sort** (array array [, int sort_flags])

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

Example 1. sort() example

```
<?php

$fruits = array ("lemon", "orange", "banana", "apple");
sort ($fruits);
reset ($fruits);
while (list ($key, $val) = each ($fruits)) {
    echo "fruits[".$key."] = ".$val."\n";
}

?>
```

This example would display:

```
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

The fruits have been sorted in alphabetical order.

The optional second parameter sort_flags may be used to modify the sorting behavior using these values:

Sorting type flags:

- SORT_REGULAR - compare items normally
- SORT_NUMERIC - compare items numerically
- SORT_STRING - compare items as strings

See also **arsort()**, **asort()**, **ksort()**, **natsort()**, **natcasesort()**, **rsort()**, **usort()**, **array_multisort()**, and **uksort()**.

> **Note:** The second parameter was added in PHP 4.

uasort

(PHP 3>= 3.0.4, PHP 4)

uasort -- Sort an array with a user-defined comparison function and maintain index association

Description

void **uasort** (array array, callback cmp_function)

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant. The comparison function is user-defined.

> **Note:** Please see **usort()** and **uksort()** for examples of user-defined comparison functions.

See also **usort()**, **uksort()**, **sort()**, **asort()**, **arsort()**, **ksort()**, and **rsort()**.

uksort

(PHP 3>= 3.0.4, PHP 4)

uksort -- Sort an array by keys using a user-defined comparison function

Description

void **uksort** (array array, callback cmp_function)

This function will sort the keys of an array using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

Example 1. uksort() example

```
function cmp ($a, $b) {
    if ($a == $b) return 0;
    return ($a > $b) ? -1 : 1;
}

$a = array (4 => "four", 3 => "three", 20 => "twenty", 10 => "ten");

uksort ($a, "cmp");

while (list ($key, $value) = each ($a)) {
    echo "$key: $value\n";
}
```

This example would display:

```
20: twenty
10: ten
4: four
3: three
```

```
3: three
```

See also **usort()**, **uasort()**, **sort()**, **asort()**, **arsort()**, **ksort()**, **natsort()**, and **rsort()**.

usort

(PHP 3>= 3.0.3, PHP 4)

usort -- Sort an array by values using a user-defined comparison function

Description

void **usort** (array array, callback cmp_function)

This function will sort an array by its values using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

> **Note:** If two members compare as equal, their order in the sorted array is undefined. Up to PHP 4.0.6 the user defined functions would keep the original order for those elements, but with the new sort algorithm intruduced with 4.1.0 this is no longer the case as there is no solution to do so in an efficient way.

Example 1. usort() example

```
function cmp ($a, $b) {
    if ($a == $b) return 0;
    return ($a > $b) ? -1 : 1;
}

$a = array (3, 2, 5, 6, 1);

usort ($a, "cmp");

while (list ($key, $value) = each ($a)) {
    echo "$key: $value\n";
}
```

This example would display:

```
0: 6
1: 5
2: 3
3: 2
4: 1
```

> **Note:** Obviously in this trivial case the **rsort()** function would be more appropriate.

Example 2. usort() example using multi-dimensional array

```
function cmp ($a, $b) {
    return strcmp($a["fruit"], $b["fruit"]);
}

$fruits[0]["fruit"] = "lemons";
$fruits[1]["fruit"] = "apples";
$fruits[2]["fruit"] = "grapes";

usort($fruits, "cmp");

while (list ($key, $value) = each ($fruits)) {
    echo "\$fruits[$key]: " . $value["fruit"] . "\n";
}
```

When sorting a multi-dimensional array, $a and $b contain references to the first index of the array.

This example would display:

```
$fruits[0]: apples
$fruits[1]: grapes
$fruits[2]: lemons
```

Example 3. usort() example using a member function of an object

```
class TestObj {
    var $name;

    function TestObj($name)
    {
        $this->name = $name;
    }

    /* This is the static comparing function: */
    function cmp_obj($a, $b)
    {
        $al = strtolower($a->name);
        $bl = strtolower($b->name);
```

```
        $al = strtolower($a->name);
        $bl = strtolower($b->name);
        if ($al == $bl) return 0;
        return ($al > $bl) ? +1 : -1;
    }
}

$a[] = new TestObj("c");
$a[] = new TestObj("b");
$a[] = new TestObj("d");

uasort($a, array ("TestObj", "cmp_obj"));

foreach ($a as $item) {
    print $item->name."\n";
}
```

This example would display:

```
b
c
d
```

See also **uasort()**, **uksort()**, **sort()**, **asort()**, **arsort()**,**ksort()**, **natsort()**, and **rsort()**.

III. Aspell functions [deprecated]

Introduction

The **aspell()** functions allows you to check the spelling on a word and offer suggestions.

Requirements

aspell works only with very old (up to .27.* or so) versions of aspell library. Neither this module, nor those versions of aspell library are supported any longer. If you want to use spell-checking capabilities in PHP, use pspell instead. It uses pspell library and works with newer versions of aspell.

Installation

You need the aspell library, available from: http://aspell.sourceforge.net/.

See Also

See also pspell.

Table of Contents

aspell_check_raw

(PHP 3>= 3.0.7, PHP 4 <= 4.2.3)

aspell_check_raw -- Check a word without changing its case or trying to trim it [deprecated]

Description

bool **aspell_check_raw** (int dictionary_link, string word)

aspell_check_raw() checks the spelling of a word, without changing its case or trying to trim it in any way and returns **TRUE** if the spelling is correct, **FALSE** if not.

Example 1. aspell_check_raw()

```
$aspell_link = aspell_new("english");

if (aspell_check_raw($aspell_link, "test")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

aspell_check

(PHP 3>= 3.0.7, PHP 4 <= 4.2.3)

aspell_check -- Check a word [deprecated]

Description

bool **aspell_check** (int dictionary_link, string word)

aspell_check() checks the spelling of a word and returns **TRUE** if the spelling is correct, **FALSE** if not.

aspell_check() checks the spelling of a word and returns **TRUE** if the spelling is correct, **FALSE** if not.

Example 1. aspell_check()

```
 $aspell_link = aspell_new("english");

 if (aspell_check($aspell_link, "testt")) {
    echo "This is a valid spelling";
 } else {
    echo "Sorry, wrong spelling";
 }
```

aspell_new

(PHP 3>= 3.0.7, PHP 4 <= 4.2.3)

aspell_new -- Load a new dictionary [deprecated]

Description

int **aspell_new** (string master [, string personal])

aspell_new() opens up a new dictionary and returns the dictionary link identifier for use in other aspell functions. Returns **FALSE** on error.

Example 1. aspell_new()

```
 $aspell_link = aspell_new("english");
```

aspell_suggest

(PHP 3>= 3.0.7, PHP 4 <= 4.2.3)

aspell_suggest -- Suggest spellings of a word [deprecated]

Description

array **aspell_suggest** (int dictionary_link, string word)

aspell_suggest() returns an array of possible spellings for the given word.

Example 1. aspell_suggest()

```
 $aspell_link = aspell_new("english");

 if (!aspell_check($aspell_link, "test")) {
    $suggestions = aspell_suggest($aspell_link, "test");

    foreach ($suggestions as $suggestion) {
       echo "Possible spelling: $suggestion<br>\n";
    }
 }
```

IV. BCMath Arbitrary Precision Mathematics Functions

Introduction

For arbitrary precision mathematics PHP offers the Binary Calculator which supports numbers of any size and precision, represented as strings.

Requirements

Since PHP 4.0.4, libbcmath is bundled with PHP. You don't need any external libraries for this extension.

Installation

In PHP 4, these functions are only available if PHP was configured with --enable-bcmath. In PHP 3, these functions are only available if PHP was NOT configured with --disable-bcmath.

The windows version of PHP has built in support for this extension. You do not need to load any additional extension in order to use these functions.

Runtime Configuration

The behaviour of these functions is affected by settings in php.ini.

Table 1. BC math configuration options

Name	Default	Changeable
bcmath.scale	0	PHP_INI_ALL

For further details and definition of the PHP_INI_* constants see **ini_set()**.

Here is a short explanation of the configuration directives.

For further details and definition of the PHP_INI_* constants see **ini_set()**.

Here is a short explanation of the configuration directives.

bcmath.scale **integer**

Number of decimal digits for all bcmath functions.

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

Table of Contents

bcadd

(PHP 3, PHP 4)

bcadd -- Add two arbitrary precision numbers

Description

string **bcadd** (string left_operand, string right_operand [, int scale])

Adds the left_operand to the right_operand and returns the sum in a string. The optional scale parameter is used to set the number of digits after the decimal place in the result.

See also **bcsub()**.

bccomp

(PHP 3, PHP 4)

bccomp -- Compare two arbitrary precision numbers

Description

int **bccomp** (string left_operand, string right_operand [, int scale])

Compares the left_operand to the right_operand and returns the result as an integer. The optional scale parameter is used to set the number of digits after the decimal place which will be used in the comparison. The return value is 0 if the two operands are equal. If the left_operand is larger than the right_operand the return value is +1 and if the left_operand is less than the right_operand the return value is -1.

bcdiv

(PHP 3, PHP 4)

bcdiv -- Divide two arbitrary precision numbers

Description

string **bcdiv** (string left_operand, string right_operand [, int scale])

Divides the left_operand by the right_operand and returns the result. The optional scale sets the number of digits after the decimal place in the result.

See also **bcmul()**.

bcmod

(PHP 3, PHP 4)

bcmod -- Get modulus of an arbitrary precision number

Description

string **bcmod** (string left_operand, string modulus)

Get the modulus of the left_operand using modulus.

See also **bcdiv()**.

bcmul

(PHP 3, PHP 4)

(PHP 3, PHP 4)

bcmul -- Multiply two arbitrary precision number

Description

string **bcmul** (string left_operand, string right_operand [, int scale])

Multiply the left_operand by the right_operand and returns the result. The optional scale sets the number of digits after the decimal place in the result.

See also **bcdiv()**.

bcpow

(PHP 3, PHP 4)

bcpow -- Raise an arbitrary precision number to another

Description

string **bcpow** (string x, string y [, int scale])

Raise x to the power y. The optional scale can be used to set the number of digits after the decimal place in the result.

See also **bcpowmod()**, and **bcsqrt()**.

bcpowmod

(PHP 5 CVS only)

bcpowmod -- Raise an arbitrary precision number to another, reduced by a specified modulus.

Description

string **bcpowmod** (string x, string y, string modulus [, int scale])

Use the fast-exponentiation method to raise x to the power y with respect to the modulus modulus. The optional scale can be used to set the number of digits after the decimal place in the result.

The following two statements are functionally identical. The **bcpowmod()** version however, executes in less time and can accept larger parameters.

```
<?php
$a = bcpowmod($x,$y,$mod);

$b = bcmod(bcpow($x,$y),$mod);

/* $a and $b are equal to each other. */
?>
```

> **Note:** Because this method uses the modulus operation, non-natural numbers may give unexpected results. A natural number is any positive non-zero integer.

See also **bcpow()**, and **bcmod()**.

bcscale

(PHP 3, PHP 4)

bcscale -- Set default scale parameter for all bc math functions

Description

string **bcscale** (int scale)

This function sets the default scale parameter for all subsequent bc math functions that do not explicitly specify a scale parameter.

bcsqrt

(PHP 3, PHP 4)

bcsqrt -- Get the square root of an arbitrary precision number

Description

string **bcsqrt** (string operand [, int scale])

Return the square root of the operand. The optional scale parameter sets the number of digits after the decimal place in the result.

See also **bcpow()**.

bcsub

bcsub

(PHP 3, PHP 4)

bcsub -- Subtract one arbitrary precision number from another

Description

string **bcsub** (string left_operand, string right_operand [, int scale])

Subtracts the right_operand from the left_operand and returns the result in a string. The optional scale parameter is used to set the number of digits after the decimal place in the result.

See also **bcadd()**.

V. Bzip2 Compression Functions

Introduction

The bzip2 functions are used to transparently read and write bzip2 (.bz2) compressed files.

Requirements

This module uses the functions of the bzip2 library by Julian Seward. This module requires bzip2/libbzip2 version >= 1.0.x.

Installation

Bzip2 support in PHP is not enabled by default. You will need to use the --with-bz2[=DIR] configuration option when compiling PHP to enable bzip2 support.

Runtime Configuration

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

Resource Types

This extension defines one resource type: a file pointer identifying the bz2-file to work on.

Predefined Constants

This extension has no constants defined.

Examples

This example opens a temporary file and writes a test string to it, then prints out the contents of the file.

Example 1. Small bzip2 Example

```
<?php

$filename = "/tmp/testfile.bz2";
$str = "This is a test string.\n";

// open file for writing
$bz = bzopen($filename, "w");

// write string to file
bzwrite($bz, $str);

// close file
bzclose($bz);

// open file for reading
$bz = bzopen($filename, "r");

// read 10 characters
print bzread($bz, 10);

// output until end of the file (or the next 1024 char) and close it.
print bzread($bz);

bzclose($bz);

?>
```

Table of Contents

bzwrite -- Binary safe bzip2 file write

bzclose

(PHP 4 >= 4.0.4)

bzclose -- Close a bzip2 file pointer

Description

int **bzclose** (resource bz)

Closes the bzip2 file referenced by the pointer bz.

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

The file pointer must be valid, and must point to a file successfully opened by **bzopen()**.

See also **bzopen()**.

bzcompress

(PHP 4 >= 4.0.4)

bzcompress -- Compress a string into bzip2 encoded data

Description

string **bzcompress** (string source [, int blocksize [, int workfactor]])

bzcompress() compresses the source string and returns it as bzip2 encoded data.

The optional parameter blocksize specifies the blocksize used during compression and should be a number from 1 to 9 with 9 giving the best compression, but using more resources to do so. blocksize defaults to 4.

The optional parameter workfactor controls how the compression phase behaves when presented with worst case, highly repetitive, input data. The value can be between 0 and 250 with 0 being a special case and 30 being the default value. Regardless of the workfactor, the generated output is the same.

Example 1. bzcompress() Example

```
<?php
$str = "sample data";
$bzstr = bzcompress($str, 9);
print( $bzstr );
?>
```

See also **bzdecompress()**.

bzdecompress

(PHP 4 >= 4.0.4)

bzdecompress -- Decompresses bzip2 encoded data

Description

string **bzdecompress** (string source [, int small])

bzdecompress() decompresses the source string containing bzip2 encoded data and returns it. If the optional parameter small is **TRUE**, an alternative decompression algorithm will be used which uses less memory (the maximum memory requirement drops to around 2300K) but works at roughly half the speed. See the bzip2 documentation for more information about this feature.

Example 1. bzdecompress()

```
<?php
$start_str = "This is not an honest face?";
$bzstr = bzcompress($start_str);

print("Compressed String: " );
print( $bzstr );
print( "\n<br>\n" );

$str = bzdecompress($bzstr);
print( "Decompressed String: " );
print( $str );
print( "\n<br>\n" );
?>
```

See also **bzcompress()**.

bzerrno

(PHP 4 >= 4.0.4)

bzerrno -- Returns a bzip2 error number

Description

int **bzerrno** (resource bz)

Description

int **bzerrno** (resource bz)

Returns the error number of any bzip2 error returned by the file pointer bz.

See also **bzerror()** and **bzerrstr()**.

bzerror

(PHP 4 >= 4.0.4)

bzerror -- Returns the bzip2 error number and error string in an array

Description

array **bzerror** (resource bz)

Returns the error number and error string, in an associative array, of any bzip2 error returned by the file pointer bz.

Example 1. bzerror() Example

```
<?php
$error = bzerror($bz);

echo $error["errno"];
echo $error["errstr"];
?>
```

See also **bzerrno()** and **bzerrstr()**.

bzerrstr

(PHP 4 >= 4.0.4)

bzerrstr -- Returns a bzip2 error string

Description

string **bzerrstr** (resource bz)

Returns the error string of any bzip2 error returned by the file pointer bz.

See also **bzerrno()** and **bzerror()**.

bzflush

(PHP 4 >= 4.0.4)

bzflush -- Force a write of all buffered data

Description

int **bzflush** (resource bz)

Forces a write of all buffered bzip2 data for the file pointer bz.

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

See also **bzread()** and **bzwrite()**.

bzopen

(PHP 4 >= 4.0.4)

bzopen -- Open a bzip2 compressed file

Description

resource **bzopen** (string filename, string mode)

Opens a bzip2 (.bz2) file for reading or writing. filename is the name of the file to open. mode is similar to the **fopen()** function (`r' for read, `w' for write, etc.).

If the open fails, the function returns **FALSE**, otherwise it returns a pointer to the newly opened file.

Example 1. bzopen() Example

```
<?php
$bz = bzopen("/tmp/foo.bz2", "r");
$decompressed_file = bzread($bz, filesize("/tmp/foo.bz2"));
bzclose($bz);

print( "The contents of /tmp/foo.bz2 are: " );
print( "\n<br>\n" );
print( $decompressed_file );
?>
```

See also **bzclose()**.

See also **bzclose()**.

bzread

(PHP 4 >= 4.0.4)

bzread -- Binary safe bzip2 file read

Description

string **bzread** (resource bz [, int length])

bzread() reads up to length bytes from the bzip2 file pointer referenced by bz. Reading stops when length (uncompressed) bytes have been read or EOF is reached, whichever comes first. If the optional parameter length is not specified, **bzread()** will read 1024 (uncompressed) bytes at a time.

Example 1. bzread() Example

```
<?php
$bz = bzopen("/tmp/foo.bz2", "r");
$str = bzread($bz, 2048);
print( $str );
?>
```

See also **bzwrite()** and **bzopen()**.

bzwrite

(PHP 4 >= 4.0.4)

bzwrite -- Binary safe bzip2 file write

Description

int **bzwrite** (resource bz, string data [, int length])

bzwrite() writes the contents of the string data to the bzip2 file stream pointed to by bz. If the optional length argument is given, writing will stop after length (uncompressed) bytes have been written or the end of string is reached, whichever comes first.

Example 1. bzwrite() Example

```
<?php
$str = "uncompressed data";
$bz = bzopen("/tmp/foo.bz2", "w");
bzwrite($bz, $str, strlen($str));
bzclose($bz);
?>
```

See also **bzread()** and **bzopen()**.

VI. Calendar functions

Introduction

The calendar extension presents a series of functions to simplify converting between different calendar formats. The intermediary or standard it is based on is the Julian Day Count. The Julian Day Count is a count of days starting from January 1st, 4713 B.C. To convert between calendar systems, you must first convert to Julian Day Count, then to the calendar system of your choice. Julian Day Count is very different from the Julian Calendar! For more information on Julian Day Count, visit http://serendipity.magnet.ch/hermetic/cal_stud/jdn.htm. For more information on calendar systems visit http://genealogy.org/~scottlee/cal-overview.html. Excerpts from this page are included in these instructions, and are in quotes.

Installation

To get these functions to work, you have to compile PHP with --enable-calendar.

The windows version of PHP has built in support for this extension. You do not need to load any additional extension in order to use these functions.

Runtime Configuration

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

Resource Types

This extension has no resource types defined.

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.

CAL_GREGORIAN (integer)

CAL_GREGORIAN (integer)

CAL_JULIAN (integer)

CAL_JEWISH (integer)

CAL_FRENCH (integer)

CAL_NUM_CALS (integer)

CAL_DOW_DAYNO (integer)

CAL_DOW_SHORT (integer)

CAL_DOW_LONG (integer)

CAL_MONTH_GREGORIAN_SHORT (integer)

CAL_MONTH_GREGORIAN_LONG (integer)

CAL_MONTH_JULIAN_SHORT (integer)

CAL_MONTH_JULIAN_LONG (integer)

CAL_MONTH_JEWISH (integer)

CAL_MONTH_FRENCH (integer)

The following constants are available since PHP 4.3.0 :

CAL_EASTER_DEFAULT (integer)

CAL_EASTER_ROMAN (integer)

CAL_EASTER_ALWAYS_GREGORIAN (integer)

CAL_EASTER_ALWAYS_JULIAN (integer)

Table of Contents

cal_days_in_month

(PHP 4 >= 4.1.0)

cal_days_in_month -- Return the number of days in a month for a given year and calendar

Description

int **cal_days_in_month** (int calendar, int month, int year)

This function will return the number of days in the month of year for the specified calendar.

See also **jdtounix()**.

cal_from_jd

(PHP 4 >= 4.1.0)

cal_from_jd -- Converts from Julian Day Count to a supported calendar and return extended information

Description

array **cal_from_jd** (int jd, int calendar)

Warning
This function is currently not documented: only the argument list is available.

cal_info

(PHP 4 >= 4.1.0)

cal_info -- Returns information about a particular calendar

Description

Description

array **cal_info** (int calendar)

Warning
This function is currently not documented; only the argument list is available.

cal_to_jd

(PHP 4 >= 4.1.0)

cal_to_jd -- Converts from a supported calendar to Julian Day Count

Description

int **cal_to_jd** (int calendar, int month, int day, int year)

Warning
This function is currently not documented; only the argument list is available.

easter_date

(PHP 3>= 3.0.9, PHP 4)

easter_date -- Get UNIX timestamp for midnight on Easter of a given year

Description

int **easter_date** ([int year])

Returns the UNIX timestamp corresponding to midnight on Easter of the given year.

Since PHP 4.3.0, the year parameter is optional and defaults to the current year according to the localtime if ommited.

Warning: This function will generate a warning if the year is outside of the range for UNIX timestamps (i.e. before 1970 or after 2037).

Example 1. easter_date() example

```
echo date ("M-d-Y", easter_date(1999));       /* "Apr-04-1999" */
echo date ("M-d-Y", easter_date(2000));       /* "Apr-23-2000" */
echo date ("M-d-Y", easter_date(2001));       /* "Apr-15-2001" */
```

The date of Easter Day was defined by the Council of Nicaea in AD325 as the Sunday after the first full moon which falls on or after the Spring Equinox. The Equinox is assumed to always fall on 21st March, so the calculation reduces to determining the date of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under the Gregorian Calendar (for years after 1753 - devised by Clavius and Lilius, and introduced by Pope Gregory XIII in October 1582, and into Britain and its then colonies in September 1752) two correction factors are added to make the cycle more accurate.

(The code is based on a C program by Simon Kershaw, <webmaster@ely.anglican.org>)

See **easter_days()** for calculating Easter before 1970 or after 2037.

easter_days

(PHP 3>= 3.0.9, PHP 4)

easter_days -- Get number of days after March 21 on which Easter falls for a given year

Description

int **easter_days** ([int year [, int method]])

Returns the number of days after March 21 on which Easter falls for a given year. If no year is specified, the current year is assumed.

Since PHP 4.3.0, the year parameter is optional and defaults to the current year according to the localtime if ommited.

The method parameter was also introduced in PHP 4.3.0 and allows to calculate easter dates based on the Gregorian calendar during the years 1582 - 1752 when set to **CAL_EASTER_ROMAN**. See the calendar constants for more valid constants.

This function can be used instead of **easter_date()** to calculate Easter for years which fall outside the range of UNIX timestamps (i.e. before 1970 or after 2037).

Example 1. easter_days() example

```
echo easter_days (1999);      /* 14, i.e. April 4  */
echo easter_days (1492);      /* 32, i.e. April 22  */
echo easter_days (1913);      /*  2, i.e. March 23  */
```

The date of Easter Day was defined by the Council of Nicaea in AD325 as the Sunday after the first full moon which falls on or after the Spring Equinox. The Equinox is assumed to always fall on 21st March, so the calculation reduces to determining the date of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under

of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under the Gregorian Calendar (for years after 1753 - devised by Clavius and Lilius, and introduced by Pope Gregory XIII in October 1582, and into Britain and its then colonies in September 1752) two correction factors are added to make the cycle more accurate.

(The code is based on a C program by Simon Kershaw, <webmaster@ely.anglican.org>)

See also **easter_date()**.

FrenchToJD

(PHP 3, PHP 4)

FrenchToJD -- Converts a date from the French Republican Calendar to a Julian Day Count

Description

int **frenchtojd** (int month, int day, int year)

Converts a date from the French Republican Calendar to a Julian Day Count.

These routines only convert dates in years 1 through 14 (Gregorian dates 22 September 1792 through 22 September 1806). This more than covers the period when the calendar was in use.

GregorianToJD

(PHP 3, PHP 4)

GregorianToJD -- Converts a Gregorian date to Julian Day Count

Description

int **gregoriantojd** (int month, int day, int year)

Valid Range for Gregorian Calendar 4714 B.C. to 9999 A.D.

Although this function can handle dates all the way back to 4714 B.C., such use may not be meaningful. The Gregorian calendar was not instituted until October 15, 1582 (or October 5, 1582 in the Julian calendar). Some countries did not accept it until much later. For example, Britain converted in 1752, The USSR in 1918 and Greece in 1923. Most European countries used the Julian calendar prior to the Gregorian.

Example 1. Calendar functions

```
<?php
$jd = GregorianToJD (10,11,1970);
echo "$jd\n";
$gregorian = JDToGregorian ($jd);
echo "$gregorian\n";
?>
```

JDDayOfWeek

(PHP 3, PHP 4)

JDDayOfWeek -- Returns the day of the week

Description

mixed **jddayofweek** (int julianday, int mode)

Returns the day of the week. Can return a string or an integer depending on the mode.

Table 1. Calendar week modes

Mode	Meaning
0	Returns the day number as an int (0=sunday, 1=monday, etc)
1	Returns string containing the day of week (english-gregorian)
2	Returns a string containing the abbreviated day of week (english-gregorian)

JDMonthName

(PHP 3, PHP 4)

JDMonthName -- Returns a month name

Description

string **jdmonthname** (int julianday, int mode)

Returns a string containing a month name. mode tells this function which calendar to convert the Julian Day Count to, and what type of month names are to be returned.

Table 1. Calendar modes

Mode	Meaning	Values
0	Gregorian - abbreviated	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
1	Gregorian	January, February, March, April, May, June, July, August, September, October, November, December

1	Gregorian	January, February, March, April, May, June, July, August, September, October, November, December
2	Julian - abbreviated	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
3	Julian	January, February, March, April, May, June, July, August, September, October, November, December
4	Jewish	Tishri, Heshvan, Kislev, Tevet, Shevat, AdarI, AdarII, Nisan, Iyyar, Sivan, Tammuz, Av, Elul
5	French Republican	Vendemiaire, Brumaire, Frimaire, Nivose, Pluviose, Ventose, Germinal, Floreal, Prairial, Messidor, Thermidor, Fructidor, Extra

JDToFrench

(PHP 3, PHP 4)

JDToFrench -- Converts a Julian Day Count to the French Republican Calendar

Description

string **jdtofrench** (int juliandaycount)

Converts a Julian Day Count to the French Republican Calendar.

JDToGregorian

(PHP 3, PHP 4)

JDToGregorian -- Converts Julian Day Count to Gregorian date

Description

string **jdtogregorian** (int julianday)

Converts Julian Day Count to a string containing the Gregorian date in the format of "month/day/year".

JDToJewish

(PHP 3, PHP 4)

JDToJewish -- Converts a Julian Day Count to the Jewish Calendar

Description

string **jdtojewish** (int julianday)

Converts a Julian Day Count the the Jewish Calendar.

JDToJulian

(PHP 3, PHP 4)

JDToJulian -- Converts a Julian Day Count to a Julian Calendar Date

Description

string **jdtojulian** (int julianday)

Converts Julian Day Count to a string containing the Julian Calendar Date in the format of "month/day/year".

jdtounix

(PHP 4)

jdtounix -- Convert Julian Day to UNIX timestamp

Description

int **jdtounix** (int jday)

This function will return a UNIX timestamp corresponding to the Julian Day given in jday or **FALSE** if jday is not inside the UNIX epoch (Gregorian years between 1970 and 2037 or 2440588 <= jday <= 2465342). The time returned is localtime (and not GMT).

See also **unixtojd()**.

JewishToJD

(PHP 3, PHP 4)

JewishToJD -- Converts a date in the Jewish Calendar to Julian Day Count

Description

int **jewishtojd** (int month, int day, int year)

Although this function can handle dates all the way back to the year 1 (3761 B.C.), such use may not be meaningful. The Jewish calendar has been in use for several thousand years, but in the early days there was no formula to determine the start of a month.

Although this function can handle dates all the way back to the year 1 (3761 B.C.), such use may not be meaningful. The Jewish calendar has been in use for several thousand years, but in the early days there was no formula to determine the start of a month. A new month was started when the new moon was first observed.

JulianToJD

(PHP 3, PHP 4)

JulianToJD -- Converts a Julian Calendar date to Julian Day Count

Description

int **juliantojd** (int month, int day, int year)

Valid Range for Julian Calendar 4713 B.C. to 9999 A.D.

Although this function can handle dates all the way back to 4713 B.C., such use may not be meaningful. The calendar was created in 46 B.C., but the details did not stabilize until at least 8 A.D., and perhaps as late at the 4th century. Also, the beginning of a year varied from one culture to another - not all accepted January as the first month.

Caution
Remember, the current calendar system being used worldwide is the Gregorian calendar. **gregoriantojd()** can be used to convert such dates to their Julian Day count.

unixtojd

(PHP 4)

unixtojd -- Convert UNIX timestamp to Julian Day

Description

int **unixtojd** ([int timestamp])

Return the Julian Day for a UNIX timestamp (seconds since 1.1.1970), or for the current day if no timestamp is given.

See also **jdtounix()**.

VII. CCVS API Functions

Introduction

These functions interface the CCVS API, allowing you to work directly with CCVS from your PHP scripts. CCVS is RedHat's solution to the "middle-man" in credit card processing. It lets you directly address the credit card clearing houses via your *nix box and a modem. Using the CCVS module for PHP, you can process credit cards directly through CCVS via your PHP Scripts. The following references will outline the process.

> **Note:** CCVS has been discontinued by Red Hat and there are no plans to issue further keys or support contracts. Those looking for a replacement can consider MCVE by Main Street Softworks as a potential replacement. It is similar in design and has documented PHP support!

Installation

To enable CCVS Support in PHP, first verify your CCVS installation directory. You will then need to configure PHP with the --with-ccvs option. If you use this option without specifying the path to your CCVS installation, PHP will attempt to look in the default CCVS Install location (/usr/local/ccvs). If CCVS is in a non-standard location, run configure with: --with-ccvs=$ccvs_path, where $ccvs_path is the path to your CCVS installation. Please note that CCVS support requires that $ccvs_path/lib and $ccvs_path/include exist, and include cv_api.h under the include directory and libccvs.a under the lib directory.

Additionally, a ccvsd process will need to be running for the configurations you intend to use in your PHP scripts. You will also need to make sure the PHP Processes are running under the same user as your CCVS was installed as (e.g. if you installed CCVS as user 'ccvs', your PHP processes must run as 'ccvs' as well.)

See Also

Additional information about CCVS can be found at http://www.redhat.com/products/ccvs. RedHat maintains slightly outdated but still useful documentation at http://www.redhat.com/products/ccvs/support/CCVS3.3docs/ProgPHP.html.

Table of Contents

ccvs_add

ccvs_add

(4.0.2 - 4.2.3 only)

ccvs_add -- Add data to a transaction

Description

string **ccvs_add** (string session, string invoice, string argtype, string argval)

Warning
This function is currently not documented; only the argument list is available.

ccvs_auth

(4.0.2 - 4.2.3 only)

ccvs_auth -- Perform credit authorization test on a transaction

Description

string **ccvs_auth** (string session, string invoice)

Warning
This function is currently not documented; only the argument list is available.

ccvs_command

(4.0.2 - 4.2.3 only)

ccvs_command -- Performs a command which is peculiar to a single protocol, and thus is not available in the general CCVS API

Description

string **ccvs_command** (string session, string type, string argval)

Warning
This function is currently not documented; only the argument list is available.

ccvs_count

(4.0.2 - 4.2.3 only)

ccvs_count -- Find out how many transactions of a given type are stored in the system

Description

int **ccvs_count** (string session, string type)

Warning
This function is currently not documented; only the argument list is available.

ccvs_delete

(4.0.2 - 4.2.3 only)

ccvs_delete -- Delete a transaction

Description

string **ccvs_delete** (string session, string invoice)

Warning
This function is currently not documented; only the argument list is available.

ccvs_done

(4.0.2 - 4.2.3 only)

ccvs_done -- Terminate CCVS engine and do cleanup work

Description

Description

string **ccvs_done** (string sess)

Warning
This function is currently not documented; only the argument list is available.

ccvs_init

(4.0.2 - 4.2.3 only)

ccvs_init -- Initialize CCVS for use

Description

string **ccvs_init** (string name)

Warning
This function is currently not documented; only the argument list is available.

ccvs_lookup

(4.0.2 - 4.2.3 only)

ccvs_lookup -- Look up an item of a particular type in the database #

Description

string **ccvs_lookup** (string session, string invoice, int inum)

Warning
This function is currently not documented; only the argument list is available.

ccvs_new

(4.0.2 - 4.2.3 only)

ccvs_new -- Create a new, blank transaction

Description

string **ccvs_new** (string session, string invoice)

Warning
This function is currently not documented; only the argument list is available.

ccvs_report

(4.0.2 - 4.2.3 only)

ccvs_report -- Return the status of the background communication process

Description

string **ccvs_report** (string session, string type)

Warning
This function is currently not documented; only the argument list is available.

ccvs_return

(4.0.2 - 4.2.3 only)

ccvs_return -- Transfer funds from the merchant to the credit card holder

Description

string **ccvs_return** (string session, string invoice)

Warning

Warning
This function is currently not documented: only the argument list is available.

ccvs_reverse

(4.0.2 - 4.2.3 only)

ccvs_reverse -- Perform a full reversal on an already-processed authorization

Description

string **ccvs_reverse** (string session, string invoice)

Warning
This function is currently not documented: only the argument list is available.

ccvs_sale

(4.0.2 - 4.2.3 only)

ccvs_sale -- Transfer funds from the credit card holder to the merchant

Description

string **ccvs_sale** (string session, string invoice)

Warning
This function is currently not documented: only the argument list is available.

ccvs_status

(4.0.2 - 4.2.3 only)

ccvs_status -- Check the status of an invoice

Description

string **ccvs_status** (string session, string invoice)

Warning
This function is currently not documented: only the argument list is available.

ccvs_textvalue

(4.0.2 - 4.2.3 only)

ccvs_textvalue -- Get text return value for previous function call

Description

string **ccvs_textvalue** (string session)

Warning
This function is currently not documented: only the argument list is available.

ccvs_void

(4.0.2 - 4.2.3 only)

ccvs_void -- Perform a full reversal on a completed transaction

Description

string **ccvs_void** (string session, string invoice)

Warning
This function is currently not documented: only the argument list is available.

VIII. COM support functions for Windows

Introduction

Introduction

COM is a technology which allows the reuse of code written in any language (by any language) using a standard calling convention and hiding behind APIs the implementation details such as what machine the Component is stored on and the executable which houses it. It can be thought of as a super Remote Procedure Call (RPC) mechanism with some basic object roots. It separates implementation from interface.

COM encourages versioning, separation of implementation from interface and hiding the implementation details such as executable location and the language it was written in.

Requirements

COM functions are only available on the Windows version of PHP.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

The windows version of PHP has built in support for this extension. You do not need to load any additional extension in order to use these functions.

Runtime Configuration

The behaviour of these functions is affected by settings in php.ini.

Table 1. Com configuration options

Name	Default	Changeable
com.allow_dcom	"0"	PHP_INI_SYSTEM
com.autoregister_typelib	"0"	PHP_INI_SYSTEM
com.autoregister_verbose	"0"	PHP_INI_SYSTEM
com.autoregister_casesensitive	"1"	PHP_INI_SYSTEM
com.typelib_file	""	PHP_INI_SYSTEM

For further details and definition of the PHP_INI_* constants see **ini_set()**.

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.

CLSCTX_INPROC_SERVER (integer)

CLSCTX_INPROC_HANDLER (integer)

CLSCTX_LOCAL_SERVER (integer)

CLSCTX_REMOTE_SERVER (integer)

CLSCTX_SERVER (integer)

CLSCTX_ALL (integer)

VT_NULL (integer)

VT_EMPTY (integer)

VT_UI1 (integer)

VT_I2 (integer)

VT_I4 (integer)

VT_R4 (integer)

VT_R8 (integer)

VT_BOOL (integer)

VT_ERROR (integer)

VT_CY (integer)

VT_DATE (integer)

VT_BSTR (integer)

VT_DECIMAL (integer)

VT_UNKNOWN (integer)

VT_DISPATCH (integer)

VT_VARIANT (integer)

VT_I1 (integer)

VT_UI2 (integer)

VT_UI4 (integer)

VT_INT (integer)

VT_UINT (integer)

VT_ARRAY (integer)

VT_UINT (integer)

VT_ARRAY (integer)

VT_BYREF (integer)

CP_ACP (integer)

CP_MACCP (integer)

CP_OEMCP (integer)

CP_UTF7 (integer)

CP_UTF8 (integer)

CP_SYMBOL (integer)

CP_THREAD_ACP (integer)

See Also

For further information on COM read the COM specification or perhaps take a look at Don Box's Yet Another COM Library (YACL)

Table of Contents

COM

(no version information, might be only in CVS)

COM -- COM class

Synopsis

```
$obj = new COM("server.object")
```

Description

The COM class provides a framework to integrate (D)COM components into your php scripts.

Methods

string **COM::COM** (string module_name [, string server_name [, int codepage]])

COM class constructor. Parameters:

module_name

: name or class-id of the requested component.

server_name

: name of the DCOM server from which the component should be fetched. If **NULL**, localhost is assumed. To allow DCOM **com.allow_dcom** has to be set to **TRUE** in php.ini.

codepage

: specifies the codepage that is used to convert php-strings to unicode-strings and vice versa. Possible values are **CP_ACP**, **CP_MACCP**, **CP_OEMCP**, **CP_SYMBOL**, **CP_THREAD_ACP**, **CP_UTF7** and **CP_UTF8**.

Example 1. COM example (1)

```
// starting word
$word = new COM("word.application") or die("Unable to instanciate Word");
print "Loaded Word, version {$word->Version}\n";

//bring it to front
$word->Visible = 1;

//open an empty document
$word->Documents->Add();

//do some weird stuff
$word->Selection->TypeText("This is a test...");
$word->Documents[1]->SaveAs("Useless test.doc");

//closing word
$word->Quit();

//free the object
$word->Release();
$word = null;
```

```
$word->Release();
$word = null;
```

Example 2. COM example (2)

```
$conn = new COM("ADODB.Connection") or die("Cannot start ADO");
$conn->Open("Provider=SQLOLEDB; Data Source=localhost;
Initial Catalog=database; User ID=user; Password=password");

$rs = $conn->Execute("SELECT * FROM sometable");    // Recordset

$num_columns = $rs->Fields->Count();
echo $num_columns . "\n";

for ($i=0; $i < $num_columns; $i++)
{
    $fld[$i] = $rs->Fields($i);
}

$rowcount = 0;
while (!$rs->EOF)
{
    for ($i=0; $i < $num_columns; $i++)
    {
        echo $fld[$i]->value . "\t";
    }
    echo "\n";
    $rowcount++;           // increments rowcount
    $rs->MoveNext();
}

$rs->Close();
$conn->Close();

$rs->Release();
$conn->Release();

$rs = null;
$conn = null;
```

VARIANT

(no version information, might be only in CVS)

VARIANT -- VARIANT class

Synopsis

```
$vVar = new VARIANT($var)
```

Description

A simple container to wrap variables into VARIANT structures.

Methods

string **VARIANT::VARIANT** ([mixed value [, int type [, int codepage]]])

VARIANT class constructor. Parameters:

value

initial value. if omitted an VT_EMPTY object is created.

type

specifies the content type of the VARIANT object. Possible values are **VT_UI1**, **VT_UI2**, **VT_UI4**, **VT_I1**, **VT_I2**, **VT_I4**, **VT_R4**, **VT_R8**, **VT_INT**, **VT_UINT**, **VT_BOOL**, **VT_ERROR**, **VT_CY**, **VT_DATE**, **VT_BSTR**, **VT_DECIMAL**, **VT_UNKNOWN**, **VT_DISPATCH** and **VT_VARIANT**. These values are mutual exclusive, but they can be combined with **VT_BYREF** to specify being a value. If omitted, the type of value is used. Consult the msdn library for additional information.

codepage

specifies the codepage that is used to convert php-strings to unicode-strings and vice versa. Possible values are **CP_ACP**, **CP_MACCP**, **CP_OEMCP**, **CP_SYMBOL**, **CP_THREAD_ACP**, **CP_UTF7** and **CP_UTF8**.

com_addref

(4.1.0 - 4.3.0 only)

com_addref -- Increases the components reference counter.

Description

void **com_addref** (void)

Increases the components reference counter.

com_get

(PHP 3>= 3.0.3, 4.0.5 - 4.3.0 only)

(PHP 3>= 3.0.3, 4.0.5 - 4.3.0 only)

com_get -- Gets the value of a COM Component's property

Description

mixed **com_get** (resource com_object, string property)

Returns the value of the property of the COM component referenced by com_object. Returns **FALSE** on error.

com_invoke

(PHP 3>= 3.0.3)

com_invoke -- Calls a COM component's method.

Description

mixed **com_invoke** (resource com_object, string function_name [, mixed function parameters, ...])

com_invoke() invokes a method of the COM component referenced by com_object. Returns **FALSE** on error, returns the function_name's return value on success.

com_isenum

(4.1.0 - 4.3.0 only)

com_isenum -- Grabs an IEnumVariant

Description

void **com_isenum** (object com_module)

Warning
This function is currently not documented: only the argument list is available.

com_load_typelib

(4.1.0 - 4.3.0 only)

com_load_typelib -- Loads a Typelib

Description

void **com_load_typelib** (string typelib_name [, int case_insensitive])

Warning
This function is currently not documented: only the argument list is available.

com_load

(PHP 3>= 3.0.3)

com_load -- Creates a new reference to a COM component

Description

string **com_load** (string module name [, string server name [, int codepage]])

com_load() creates a new COM component and returns a reference to it. Returns **FALSE** on failure. Possible values for codepage are **CP_ACP**, **CP_MACCP**, **CP_OEMCP**, **CP_SYMBOL**, **CP_THREAD_ACP**, **CP_UTF7** and **CP_UTF8**.

com_propget

(PHP 3>= 3.0.3, 4.0.5 - 4.3.0 only)

com_propget -- Gets the value of a COM Component's property

Description

mixed **com_propget** (resource com_object, string property)

This function is an alias for **com_get()**.

com_propput

(PHP 3>= 3.0.3, 4.0.5 - 4.3.0 only)

com_propput -- Assigns a value to a COM component's property

Description

Description

void **com_propput** (resource com_object, string property, mixed value)

This function is an alias for **com_set()**.

com_propset

(PHP 3>= 3.0.3, 4.0.5 - 4.3.0 only)

com_propset -- Assigns a value to a COM component's property

Description

void **com_propset** (resource com_object, string property, mixed value)

This function is an alias for **com_set()**.

com_release

(4.1.0 - 4.3.0 only)

com_release -- Decreases the components reference counter.

Description

void **com_release** (void)

Decreases the components reference counter.

com_set

(PHP 3>= 3.0.3, 4.0.5 - 4.3.0 only)

com_set -- Assigns a value to a COM component's property

Description

void **com_set** (resource com_object, string property, mixed value)

Sets the value of the property of the COM component referenced by com_object. Returns the newly set value if succeeded, **FALSE** on error.

IX. Class/Object Functions

Introduction

These functions allow you to obtain information about classes and instance objects. You can obtain the name of the class to which a object belongs, as well as its member properties and methods. Using these functions, you can find out not only the class membership of an object, but also its parentage (i.e. what class is the object class extending).

Requirements

No external libraries are needed to build this extension.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

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

In this example, we first define a base class and an extension of the class. The base class describes a general vegetable, whether it is edible or not and what is its color. The subclass Spinach adds a method to cook it and another to find out if it is cooked.

Example 1. classes.inc

```
<?php

// base class with member properties and methods
```

```
.php

// base class with member properties and methods
class Vegetable {

    var $edible;
    var $color;

    function Vegetable( $edible, $color="green" ) {
        $this->edible = $edible;
        $this->color =  $color;
    }

    function is_edible() {
        return $this->edible;
    }

    function what_color() {
        return $this->color;
    }

} // end of class Vegetable

// extends the base class
class Spinach extends Vegetable {

    var $cooked = false;

    function Spinach() {
        $this->Vegetable( true, "green" );
    }

    function cook_it() {
        $this->cooked = true;
    }

    function is_cooked() {
        return $this->cooked;
    }

} // end of class Spinach

?>
```

We then instantiate 2 objects from these classes and print out information about them, including their class parentage. We also define some utility functions, mainly to have a nice printout of the variables.

Example 2. test_script.php

```
<pre>
<?php

include "classes.inc";

// utility functions

function print_vars($obj) {
    $arr = get_object_vars($obj);
    while (list($prop, $val) = each($arr))
        echo "\t$prop = $val\n";
}

function print_methods($obj) {
    $arr = get_class_methods(get_class($obj));
    foreach ($arr as $method)
        echo "\tfunction $method()\n";
}

function class_parentage($obj, $class) {
    global $$obj;
    if (is_subclass_of($$obj, $class)) {
        echo "Object $obj belongs to class ".get_class($$obj);
        echo " a subclass of $class\n";
    } else {
        echo "Object $obj does not belong to a subclass of $class\n";
    }
}

// instantiate 2 objects

$veggie = new Vegetable(true,"blue");
$leafy = new Spinach();

// print out information about objects
echo "veggie: CLASS ".get_class($veggie)."\n";
echo "leafy: CLASS ".get_class($leafy);
echo ", PARENT ".get_parent_class($leafy)."\n";

// show veggie properties
echo "\nveggie: Properties\n";
print_vars($veggie);

// and leafy methods
echo "\nleafy: Methods\n";
print_methods($leafy);
```

```
echo "\nleafy: Methods\n";
print_methods($leafy);

echo "\nParentage:\n";
class_parentage("leafy", "Spinach");
class_parentage("leafy", "Vegetable");
?>
</pre>
```

One important thing to note in the example above is that the object $leafy is an instance of the class **Spinach** which is a subclass of **Vegetable**, therefore the last part of the script above will output:

```
[...]
Parentage:
Object leafy does not belong to a subclass of Spinach
Object leafy belongs to class spinach a subclass of Vegetable
```

Table of Contents

call_user_method_array

(PHP 4 >= 4.0.5)

call_user_method_array -- Call a user method given with an array of parameters [deprecated]

Description

mixed **call_user_method_array** (string method_name, object obj [, array paramarr])

Warning
The **call_user_method_array()** function is deprecated as of PHP 4.1.0, use the **call_user_func_array()** variety with the array(&$obj, "method_name") syntax instead.

Calls the method referred by method_name from the user defined obj object, using the parameters in paramarr.

See also: **call_user_func_array()**, **call_user_func()**, **call_user_method()**.

> **Note:** This function was added to the CVS code after release of PHP 4.0.4pl1

call_user_method

(PHP 3>= 3.0.3, PHP 4)

call_user_method -- Call a user method on an specific object [deprecated]

Description

mixed **call_user_method** (string method_name, object obj [, mixed parameter [, mixed ...]])

Warning
The **call_user_method()** function is deprecated as of PHP 4.1.0, use the **call_user_func()** variety with the array(&$obj, "method_name") syntax instead.

Calls the method referred by method_name from the user defined obj object. An example of usage is below, where we define a class, instantiate an object and use **call_user_method()** to call indirectly its print_info method.

```
<?php
class Country {
    var $NAME;
    var $TLD;

    function Country($name, $tld) {
        $this->NAME = $name;
        $this->TLD = $tld;
    }

    function print_info($prestr="") {
        echo $prestr."Country: ".$this->NAME."\n";
        echo $prestr."Top Level Domain: ".$this->TLD."\n";
    }
}

$cntry = new Country("Peru","pe");
```

```
$cntry = new Country("Peru","pe");

echo "* Calling the object method directly\n";
$cntry->print_info();

echo "\n* Calling the same method indirectly\n";
call_user_method ("print_info", $cntry, "\t");
?>
```

See also **call_user_func_array()**, **call_user_func()**, and **call_user_method_array()**.

class_exists

(PHP 4)

class_exists -- Checks if the class has been defined

Description

bool **class_exists** (string class_name)

This function returns **TRUE** if the class given by class_name has been defined, **FALSE** otherwise.

See also **get_declared_classes()**.

get_class_methods

(PHP 4)

get_class_methods -- Returns an array of class methods' names

Description

array **get_class_methods** (mixed class_name)

This function returns an array of method names defined for the class specified by class_name.

> **Note:** As of PHP 4.0.6, you can specify the object itself instead of class_name. For example:
>
> ```
> $class_methods = get_class_methods($my_class); // see below the full example
> ```

Example 1. get_class_methods() example

```
<?php

class myclass {
    // constructor
    function myclass() {
        return(TRUE);
    }

    // method 1
    function myfunc1() {
        return(TRUE);
    }

    // method 2
    function myfunc2() {
        return(TRUE);
    }
}

$my_object = new myclass();

$class_methods = get_class_methods(get_class($my_object));

foreach ($class_methods as $method_name) {
    echo "$method_name\n";
}

?>
```

Will produce:

```
myclass
myfunc1
myfunc2
```

See also **get_class_vars()** and **get_object_vars()**.

get_class_vars

(PHP 4)

get_class_vars -- Returns an array of default properties of the class

(PHP 4)

get_class_vars -- Returns an array of default properties of the class

Description

array **get_class_vars** (string class_name)

This function will return an associative array of default properties of the class. The resulting array elements are in the form of varname => value.

> **Note:** Prior to PHP 4.2.0, Uninitialized class variables will not be reported by **get_class_vars()**.

Example 1. get_class_vars() example

```
<?php

class myclass {

    var $var1; // this has no default value...
    var $var2 = "xyz";
    var $var3 = 100;

    // constructor
    function myclass() {
        return(TRUE);
    }

}

$my_class = new myclass();

$class_vars = get_class_vars(get_class($my_class));

foreach ($class_vars as $name => $value) {
    echo "$name : $value\n";
}

?>
```

Will produce:

```
// Before PHP 4.2.0
var2 : xyz
var3 : 100

// As of PHP 4.2.0
var1 :
var2 : xyz
var3 : 100
```

See also **get_class_methods()**, **get_object_vars()**

get_class

(PHP 4)

get_class -- Returns the name of the class of an object

Description

string **get_class** (object obj)

This function returns the name of the class of which the object obj is an instance. Returns **FALSE** if obj is not an object.

> **Note:** **get_class()** returns a user defined class name in lowercase. A class defined in a PHP extension is returned in its original notation.

See also **get_parent_class()**, **gettype()**, and **is_subclass_of()**.

get_declared_classes

(PHP 4)

get_declared_classes -- Returns an array with the name of the defined classes

Description

array **get_declared_classes** (void)

This function returns an array of the names of the declared classes in the current script.

> **Note:** In PHP 4.0.1pl2, three extra classes are returned at the beginning of the array: **stdClass** (defined in Zend/zend.c), **OverloadedTestClass** (defined in ext/standard/basic_functions.c) and **Directory** (defined in ext/standard/dir.c).
>
> Also note that depending on what libraries you have compiled into PHP, additional classes could be present. This means that you will not be able to define your own classes using these names. There is a list of predefined classes in the Predefined Classes section of the appendices.

Also note that depending on what libraries you have compiled into PHP, additional classes could be present. This means that you will not be able to define your own classes using these names. There is a list of predefined classes in the Predefined Classes section of the appendices.

See also **class_exists()**.

get_object_vars

(PHP 4)

get_object_vars -- Returns an associative array of object properties

Description

array **get_object_vars** (object obj)

This function returns an associative array of defined object properties for the specified object obj.

> **Note:** In versions prior to PHP 4.2.0, if the variables declared in the class of which the obj is an instance, have not been assigned a value, those will not be returned in the array. In versions after PHP 4.2.0, the key will be assigned with a **NULL** value.

Example 1. Use of get_object_vars()

```
<?php
class Point2D {
    var $x, $y;
    var $label;

    function Point2D($x, $y) {
        $this->x = $x;
        $this->y = $y;
    }

    function setLabel($label) {
        $this->label = $label;
    }

    function getPoint() {
        return array("x" => $this->x,
                     "y" => $this->y,
                     "label" => $this->label);
    }
}

// "$label" is declared but not defined
$p1 = new Point2D(1.233, 3.445);
print_r(get_object_vars($p1));

$p1->setLabel("point #1");
print_r(get_object_vars($p1));

?>
```

The printout of the above program will be:

```
Array
(
    [x] => 1.233
    [y] => 3.445
    [label] =>
)

Array
(
    [x] => 1.233
    [y] => 3.445
    [label] => point #1
)
```

See also **get_class_methods()** and **get_class_vars()**!

get_parent_class

(PHP 4)

get_parent_class -- Retrieves the parent class name for object or class

Description

string **get_parent_class** (mixed obj)

If obj is an object, returns the name of the parent class of the class of which obj is an instance.

If obj is a string, returns the name of the parent class of the class with that name. This functionality was added in PHP 4.0.5.

See also **get_class()** and **is_subclass_of()**

is_a

(PHP 4 >= 4.2.0)

is_a -- Returns **TRUE** if the object is of this class or has this class as one of its parents

(PHP 4 >= 4.2.0)

is_a -- Returns **TRUE** if the object is of this class or has this class as one of its parents

Description

bool **is_a** (object object, string class_name)

This function returns **TRUE** if the object is of this class or has this class as one of its parents, **FALSE** otherwise.

See also **get_class()**, **get_parent_class()**, and **is_subclass_of()**.

is_subclass_of

(PHP 4)

is_subclass_of -- Returns **TRUE** if the object has this class as one of its parents

Description

bool **is_subclass_of** (object object, string class_name)

This function returns **TRUE** if the object object, belongs to a class which is a subclass of class_name, **FALSE** otherwise.

See also **get_class()**, **get_parent_class()** and **is_a()**.

method_exists

(PHP 4)

method_exists -- Checks if the class method exists

Description

bool **method_exists** (object object, string method_name)

This function returns **TRUE** if the method given by method_name has been defined for the given object, **FALSE** otherwise.

X. ClibPDF functions

Introduction

ClibPDF lets you create PDF documents with PHP. ClibPDF functionality and API are similar to PDFlib. This documentation should be read alongside the ClibPDF manual since it explains the library in much greater detail.

Many functions in the native ClibPDF and the PHP module, as well as in PDFlib, have the same name. All functions except for **cpdf_open()** take the handle for the document as their first parameter.

Currently this handle is not used internally since ClibPDF does not support the creation of several PDF documents at the same time. Actually, you should not even try it, the results are unpredictable. I can't oversee what the consequences in a multi threaded environment are. According to the author of ClibPDF this will change in one of the next releases (current version when this was written is 1.10). If you need this functionality use the pdflib module.

A nice feature of ClibPDF (and PDFlib) is the ability to create the pdf document completely in memory without using temporary files. It also provides the ability to pass coordinates in a predefined unit length. (This feature can also be simulated by **pdf_translate()** when using the PDFlib functions.)

Another nice feature of ClibPDF is the fact that any page can be modified at any time even if a new page has been already opened. The function **cpdf_set_current_page()** allows to leave the current page and presume modifying an other page.

Most of the functions are fairly easy to use. The most difficult part is probably creating a very simple PDF document at all. The following example should help you to get started. It creates a document with one page. The page contains the text "Times-Roman" in an outlined 30pt font. The text is underlined.

Requirements

In order to use the ClibPDF functions you need to install the ClibPDF package. It is available for download from FastIO, but requires that you purchase a license for commercial use. PHP requires that you use cpdflib >= 2.

Installation

To get these functions to work, you have to compile PHP with --with-cpdflib[=DIR]. DIR is the cpdflib install directory, defaults to /usr. In addition you can specify the jpeg library and the tiff library for ClibPDF to use. To do so add to your configure line the options --with-jpeg-dir[=DIR] --with-tiff-dir[=DIR].

Runtime Configuration

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

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.

CPDF_PM_NONE (integer)

CPDF_PM_OUTLINES (integer)

CPDF_PM_THUMBS (integer)

CPDF_PM_THUMBS (integer)

CPDF_PM_FULLSCREEN (integer)

CPDF_PL_SINGLE (integer)

CPDF_PL_1COLUMN (integer)

CPDF_PL_2LCOLUMN (integer)

CPDF_PL_2RCOLUMN (integer)

Examples

Example 1. Simple ClibPDF Example

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842, 1.0);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
cpdf_begin_text($cpdf);
cpdf_set_font($cpdf, "Times-Roman", 30, "WinAnsiEncoding");
cpdf_set_text_rendering($cpdf, 1);
cpdf_text($cpdf, "Times Roman outlined", 50, 750);
cpdf_end_text($cpdf);
cpdf_moveto($cpdf, 50, 740);
cpdf_lineto($cpdf, 330, 740);
cpdf_stroke($cpdf);
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

The pdflib distribution contains a more complex example which creates a series of pages with an analog clock. Here is that example converted into PHP using the ClibPDF extension:

Example 2. pdfclock example from pdflib 2.0 distribution

```
<?php
$radius = 200;
$margin = 20;
$pagecount = 40;

$pdf = cpdf_open(0);
cpdf_set_creator($pdf, "pdf_clock.php3");
cpdf_set_title($pdf, "Analog Clock");

while($pagecount-- > 0) {
  cpdf_page_init($pdf, $pagecount+1, 0, 2 * ($radius + $margin), 2 * ($radius + $margin), 1.0);

  cpdf_set_page_animation($pdf, 4, 0.5, 0, 0, 0);  /* wipe */

  cpdf_translate($pdf, $radius + $margin, $radius + $margin);
  cpdf_save($pdf);
  cpdf_setrgbcolor($pdf, 0.0, 0.0, 1.0);

  /* minute strokes */
  cpdf_setlinewidth($pdf, 2.0);
  for ($alpha = 0; $alpha < 360; $alpha += 6)
   {
    cpdf_rotate($pdf, 6.0);
    cpdf_moveto($pdf, $radius, 0.0);
    cpdf_lineto($pdf, $radius-$margin/3, 0.0);
    cpdf_stroke($pdf);
   }

  cpdf_restore($pdf);
  cpdf_save($pdf);

  /* 5 minute strokes */
  cpdf_setlinewidth($pdf, 3.0);
  for ($alpha = 0; $alpha < 360; $alpha += 30)
  {
    cpdf_rotate($pdf, 30.0);
    cpdf_moveto($pdf, $radius, 0.0);
    cpdf_lineto($pdf, $radius-$margin, 0.0);
    cpdf_stroke($pdf);
  }

  $ltime = getdate();

  /* draw hour hand */
  cpdf_save($pdf);
  cpdf_rotate($pdf, -(($ltime['minutes']/60.0) + $ltime['hours'] - 3.0) * 30.0);
  cpdf_moveto($pdf, -$radius/10, -$radius/20);
  cpdf_lineto($pdf, $radius/2, 0.0);
  cpdf_lineto($pdf, -$radius/10, $radius/20);
  cpdf_closepath($pdf);
  cpdf_fill($pdf);
  cpdf_restore($pdf);

  /* draw minute hand */
```

```
    /* draw minute hand */
    cpdf_save($pdf);
    cpdf_rotate($pdf, -(($ltime['seconds']/60.0) + $ltime['minutes'] - 15.0) * 6.0);
    cpdf_moveto($pdf, -$radius/10, -$radius/20);
    cpdf_lineto($pdf, $radius * 0.8, 0.0);
    cpdf_lineto($pdf, -$radius/10, $radius/20);
    cpdf_closepath($pdf);
    cpdf_fill($pdf);
    cpdf_restore($pdf);

    /* draw second hand */
    cpdf_setrgbcolor($pdf, 1.0, 0.0, 0.0);
    cpdf_setlinewidth($pdf, 2);
    cpdf_save($pdf);
    cpdf_rotate($pdf, -(($ltime['seconds'] - 15.0) * 6.0));
    cpdf_moveto($pdf, -$radius/5, 0.0);
    cpdf_lineto($pdf, $radius, 0.0);
    cpdf_stroke($pdf);
    cpdf_restore($pdf);

    /* draw little circle at center */
    cpdf_circle($pdf, 0, 0, $radius/30);
    cpdf_fill($pdf);

    cpdf_restore($pdf);

    cpdf_finalize_page($pdf, $pagecount+1);
  }

  cpdf_finalize($pdf);
  Header("Content-type: application/pdf");
  cpdf_output_buffer($pdf);
  cpdf_close($pdf);
  ?>
```

See Also

See also the PDFlib extension documentation.

Table of Contents

cpdf_set_viewer_preferences -- How to show the document in the viewer
cpdf_set_word_spacing -- Sets spacing between words
cpdf_setdash -- Sets dash pattern
cpdf_setflat -- Sets flatness
cpdf_setgray_fill -- Sets filling color to gray value
cpdf_setgray_stroke -- Sets drawing color to gray value
cpdf_setgray -- Sets drawing and filling color to gray value
cpdf_setlinecap -- Sets linecap parameter
cpdf_setlinejoin -- Sets linejoin parameter
cpdf_setlinewidth -- Sets line width
cpdf_setmiterlimit -- Sets miter limit
cpdf_setrgbcolor_fill -- Sets filling color to rgb color value
cpdf_setrgbcolor_stroke -- Sets drawing color to rgb color value
cpdf_setrgbcolor -- Sets drawing and filling color to rgb color value
cpdf_show_xy -- Output text at position
cpdf_show -- Output text at current position
cpdf_stringwidth -- Returns width of text in current font
cpdf_stroke -- Draw line along path
cpdf_text -- Output text with parameters
cpdf_translate -- Sets origin of coordinate system

cpdf_add_annotation

(PHP 3>= 3.0.12, PHP 4)

cpdf_add_annotation -- Adds annotation

Description

void **cpdf_add_annotation** (int pdf document, float llx, float lly, float urx, float ury, string title, string content [, int mode])

The **cpdf_add_annotation()** adds a note with the lower left corner at (llx, lly) and the upper right corner at (urx, ury).

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

cpdf_add_outline

(PHP 3>= 3.0.9, PHP 4)

cpdf_add_outline -- Adds bookmark for current page

Description

void **cpdf_add_outline** (int pdf document, string text)

The **cpdf_add_outline()** function adds a bookmark with text text that points to the current page.

Example 1. Adding a page outline

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
// ...
// some drawing
// ...
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

cpdf_arc

(PHP 3>= 3.0.8, PHP 4)

cpdf_arc -- Draws an arc

Description

void **cpdf_arc** (int pdf document, float x-coor, float y-coor, float radius, float start, float end [, int mode])

The **cpdf_arc()** function draws an arc with center at point (x-coor, y-coor) and radius radius, starting at angle start and ending at angle end.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_circle()**.

cpdf_begin_text

(PHP 3>= 3.0.8, PHP 4)

cpdf_begin_text -- Starts text section

Description

void **cpdf_begin_text** (int pdf document)

void **cpdf_begin_text** (int pdf document)

The **cpdf_begin_text()** function starts a text section. It must be ended with **cpdf_end_text()**.

Example 1. Text output

```
<?php
cpdf_begin_text($pdf);
cpdf_set_font($pdf, 16, "Helvetica", "WinAnsiEncoding");
cpdf_text($pdf, 100, 100, "Some text");
cpdf_end_text($pdf)
?>
```

See also **cpdf_end_text()**.

cpdf_circle

(PHP 3>= 3.0.8, PHP 4)

cpdf_circle -- Draw a circle

Description

void **cpdf_circle** (int pdf document, float x-coor, float y-coor, float radius [, int mode])

The **cpdf_circle()** function draws a circle with center at point (x-coor, y-coor) and radius radius.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_arc()**.

cpdf_clip

(PHP 3>= 3.0.8, PHP 4)

cpdf_clip -- Clips to current path

Description

void **cpdf_clip** (int pdf document)

The **cpdf_clip()** function clips all drawing to the current path.

cpdf_close

(PHP 3>= 3.0.8, PHP 4)

cpdf_close -- Closes the pdf document

Description

void **cpdf_close** (int pdf document)

The **cpdf_close()** function closes the pdf document. This should be the last function even after **cpdf_finalize()**, **cpdf_output_buffer()** and **cpdf_save_to_file()**.

See also **cpdf_open()**.

cpdf_closepath_fill_stroke

(PHP 3>= 3.0.8, PHP 4)

cpdf_closepath_fill_stroke -- Close, fill and stroke current path

Description

void **cpdf_closepath_fill_stroke** (int pdf document)

The **cpdf_closepath_fill_stroke()** function closes, fills the interior of the current path with the current fill color and draws current path.

See also **cpdf_closepath()**, **cpdf_stroke()**, **cpdf_fill()**, **cpdf_setgray_fill()**, **cpdf_setgray()**, **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_closepath_stroke

(PHP 3>= 3.0.8, PHP 4)

cpdf_closepath_stroke -- Close path and draw line along path

Description

void **cpdf_closepath_stroke** (int pdf document)

The **cpdf_closepath_stroke()** function is a combination of **cpdf_closepath()** and **cpdf_stroke()**. Then clears the path.

See also **cpdf_closepath()**, **cpdf_stroke()**.

See also **cpdf_closepath()**, **cpdf_stroke()**.

cpdf_closepath

(PHP 3>= 3.0.8, PHP 4)

cpdf_closepath -- Close path

Description

void **cpdf_closepath** (int pdf document)

The **cpdf_closepath()** function closes the current path.

cpdf_continue_text

(PHP 3>= 3.0.8, PHP 4)

cpdf_continue_text -- Output text in next line

Description

void **cpdf_continue_text** (int pdf document, string text)

The **cpdf_continue_text()** function outputs the string in text in the next line.

See also **cpdf_show_xy()**, **cpdf_text()**, **cpdf_set_leading()**, **cpdf_set_text_pos()**.

cpdf_curveto

(PHP 3>= 3.0.8, PHP 4)

cpdf_curveto -- Draws a curve

Description

void **cpdf_curveto** (int pdf document, float x1, float y1, float x2, float y2, float x3, float y3 [, int mode])

The **cpdf_curveto()** function draws a Bezier curve from the current point to the point (x3, y3) using (x1, y1) and (x2, y2) as control points.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_rlineto()**, **cpdf_lineto()**.

cpdf_end_text

(PHP 3>= 3.0.8, PHP 4)

cpdf_end_text -- Ends text section

Description

void **cpdf_end_text** (int pdf document)

The **cpdf_end_text()** function ends a text section which was started with **cpdf_begin_text()**.

Example 1. Text output

```
<?php
cpdf_begin_text($pdf);
cpdf_set_font($pdf, 16, "Helvetica", "WinAnsiEncoding");
cpdf_text($pdf, 100, 100, "Some text");
cpdf_end_text($pdf)
?>
```

See also **cpdf_begin_text()**.

cpdf_fill_stroke

(PHP 3>= 3.0.8, PHP 4)

cpdf_fill_stroke -- Fill and stroke current path

Description

void **cpdf_fill_stroke** (int pdf document)

The **cpdf_fill_stroke()** function fills the interior of the current path with the current fill color and draws current path.

See also **cpdf_closepath()**, **cpdf_stroke()**, **cpdf_fill()**, **cpdf_setgray_fill()**, **cpdf_setgray()**, **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_fill

(PHP 3>= 3.0.8, PHP 4)

cpdf_fill -- Fill current path

(PHP 3>= 3.0.8, PHP 4)

cpdf_fill -- Fill current path

Description

void **cpdf_fill** (int pdf document)

The **cpdf_fill()** function fills the interior of the current path with the current fill color.

See also **cpdf_closepath()**, **cpdf_stroke()**, **cpdf_setgray_fill()**, **cpdf_setgray()**, **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_finalize_page

(PHP 3>= 3.0.10, PHP 4)

cpdf_finalize_page -- Ends page

Description

void **cpdf_finalize_page** (int pdf document, int page number)

The **cpdf_finalize_page()** function ends the page with page number page number.

This function is only for saving memory. A finalized page takes less memory but cannot be modified anymore.

See also **cpdf_page_init()**.

cpdf_finalize

(PHP 3>= 3.0.8, PHP 4)

cpdf_finalize -- Ends document

Description

void **cpdf_finalize** (int pdf document)

The **cpdf_finalize()** function ends the document. You still have to call **cpdf_close()**

See also **cpdf_close()**.

cpdf_global_set_document_limits

(PHP 4)

cpdf_global_set_document_limits -- Sets document limits for any pdf document

Description

void **cpdf_global_set_document_limits** (int maxpages, int maxfonts, int maximages, int maxannotations, int maxobjects)

The **cpdf_global_set_document_limits()** function sets several document limits. This function has to be called before **cpdf_open()** to take effect. It sets the limits for any document open afterwards.

See also **cpdf_open()**.

cpdf_import_jpeg

(PHP 3>= 3.0.9, PHP 4)

cpdf_import_jpeg -- Opens a JPEG image

Description

int **cpdf_import_jpeg** (int pdf document, string file name, float x-coor, float y-coor, float angle, float width, float height, float x-scale, float y-scale [, int mode])

The **cpdf_import_jpeg()** function opens an image stored in the file with the name file name. The format of the image has to be jpeg. The image is placed on the current page at position (x-coor, y-coor). The image is rotated by angle degrees.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_place_inline_image()**.

cpdf_lineto

(PHP 3>= 3.0.8, PHP 4)

cpdf_lineto -- Draws a line

Description

void **cpdf_lineto** (int pdf document, float x-coor, float y-coor [, int mode])

The **cpdf_lineto()** function draws a line from the current point to the point with coordinates (x-coor, y-coor).

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

cpdf_moveto

(PHP 3>= 3.0.8, PHP 4)

cpdf_moveto -- Sets current point

Description

void **cpdf_moveto** (int pdf document, float x-coor, float y-coor [, int mode])

The **cpdf_moveto()** function set the current point to the coordinates x-coor and y-coor.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

cpdf_newpath

(PHP 3>= 3.0.9, PHP 4)

cpdf_newpath -- Starts a new path

Description

void **cpdf_newpath** (int pdf document)

The **cpdf_newpath()** starts a new path on the document given by the pdf document parameter.

cpdf_open

(PHP 3>= 3.0.8, PHP 4)

cpdf_open -- Opens a new pdf document

Description

int **cpdf_open** (int compression [, string filename])

The **cpdf_open()** function opens a new pdf document. The first parameter turns document compression on if it is unequal to 0. The second optional parameter sets the file in which the document is written. If it is omitted the document is created in memory and can either be written into a file with the **cpdf_save_to_file()** or written to standard output with **cpdf_output_buffer()**.

> **Note:** The return value will be needed in further versions of ClibPDF as the first parameter in all other functions which are writing to the pdf document.
>
> The ClibPDF library takes the filename "-" as a synonym for stdout. If PHP is compiled as an apache module this will not work because the way ClibPDF outputs to stdout does not work with apache. You can solve this problem by skipping the filename and using **cpdf_output_buffer()** to output the pdf document.

See also **cpdf_close()**, **cpdf_output_buffer()**.

cpdf_output_buffer

(PHP 3>= 3.0.9, PHP 4)

cpdf_output_buffer -- Outputs the pdf document in memory buffer

Description

void **cpdf_output_buffer** (int pdf document)

The **cpdf_output_buffer()** function outputs the pdf document to stdout. The document has to be created in memory which is the case if **cpdf_open()** has been called with no filename parameter.

See also **cpdf_open()**.

cpdf_page_init

(PHP 3>= 3.0.8, PHP 4)

cpdf_page_init -- Starts new page

Description

void **cpdf_page_init** (int pdf document, int page number, int orientation, float height, float width [, float unit])

The **cpdf_page_init()** function starts a new page with height height and width width. The page has number page number and orientation orientation. orientation can be 0 for portrait and 1 for landscape. The last optional parameter unit sets the unit for the coordinate system. The value should be the number of postscript points per unit. Since one inch is equal to 72 points, a value of 72 would set the unit to one inch. The default is also 72.

See also **cpdf_set_current_page()**.

cpdf_place_inline_image

(PHP 3>= 3.0.9, PHP 4)

(PHP 3>= 3.0.9, PHP 4)

cpdf_place_inline_image -- Places an image on the page

Description

void **cpdf_place_inline_image** (int pdf document, int image, float x-coor, float y-coor, float angle, float width, float height [, int mode])

The **cpdf_place_inline_image()** function places an image created with the php image functions on the page at position (x-coor, y-coor). The image can be scaled at the same time.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_import_jpeg()**.

cpdf_rect

(PHP 3>= 3.0.8, PHP 4)

cpdf_rect -- Draw a rectangle

Description

void **cpdf_rect** (int pdf document, float x-coor, float y-coor, float width, float height [, int mode])

The **cpdf_rect()** function draws a rectangle with its lower left corner at point (x-coor, y-coor). This width is set to width. This height is set to height.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

cpdf_restore

(PHP 3>= 3.0.8, PHP 4)

cpdf_restore -- Restores formerly saved environment

Description

void **cpdf_restore** (int pdf document)

The **cpdf_restore()** function restores the environment saved with **cpdf_save()**. It works like the postscript command grestore. Very useful if you want to translate or rotate an object without effecting other objects.

Example 1. Save/Restore

```
<?php
cpdf_save($pdf);
// do all kinds of rotations, transformations, ...
cpdf_restore($pdf)
?>
```

See also **cpdf_save()**.

cpdf_rlineto

(PHP 3>= 3.0.9, PHP 4)

cpdf_rlineto -- Draws a line

Description

void **cpdf_rlineto** (int pdf document, float x-coor, float y-coor [, int mode])

The **cpdf_rlineto()** function draws a line from the current point to the relative point with coordinates (x-coor, y-coor).

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

cpdf_rmoveto

(PHP 3>= 3.0.9, PHP 4)

cpdf_rmoveto -- Sets current point

Description

void **cpdf_rmoveto** (int pdf document, float x-coor, float y-coor [, int mode])

The **cpdf_rmoveto()** function set the current point relative to the coordinates x-coor and y-coor.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**.

cpdf_rotate_text

See also **cpdf_moveto()**.

cpdf_rotate_text

(PHP 3>= 3.0.9, PHP 4)

cpdf_rotate_text -- Sets text rotation angle

Description

void **cpdf_rotate_text** (int pdfdoc, float angle)

Warning
This function is currently not documented: only the argument list is available.

cpdf_rotate

(PHP 3>= 3.0.8, PHP 4)

cpdf_rotate -- Sets rotation

Description

void **cpdf_rotate** (int pdf document, float angle)

The **cpdf_rotate()** function set the rotation in degrees to angle.

cpdf_save_to_file

(PHP 3>= 3.0.8, PHP 4)

cpdf_save_to_file -- Writes the pdf document into a file

Description

void **cpdf_save_to_file** (int pdf document, string filename)

The **cpdf_save_to_file()** function outputs the pdf document into a file if it has been created in memory.

This function is not needed if the pdf document has been open by specifying a filename as a parameter of **cpdf_open()**.

See also **cpdf_output_buffer()**, **cpdf_open()**.

cpdf_save

(PHP 3>= 3.0.8, PHP 4)

cpdf_save -- Saves current environment

Description

void **cpdf_save** (int pdf document)

The **cpdf_save()** function saves the current environment. It works like the postscript command gsave. Very useful if you want to translate or rotate an object without effecting other objects.

See also **cpdf_restore()**.

cpdf_scale

(PHP 3>= 3.0.8, PHP 4)

cpdf_scale -- Sets scaling

Description

void **cpdf_scale** (int pdf document, float x-scale, float y-scale)

The **cpdf_scale()** function set the scaling factor in both directions.

cpdf_set_action_url

(PHP 3>= 3.0.9, PHP 4)

cpdf_set_action_url -- Sets hyperlink

Description

void **cpdf_set_action_url** (int pdfdoc, float xll, float yll, float xur, float xur, string url [, int mode])

Warning
This function is currently not documented: only the argument list is available.

Warning
This function is currently not documented: only the argument list is available.

cpdf_set_char_spacing

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_char_spacing -- Sets character spacing

Description

void **cpdf_set_char_spacing** (int pdf document, float space)

The **cpdf_set_char_spacing()** function sets the spacing between characters.

See also **cpdf_set_word_spacing()**, **cpdf_set_leading()**.

cpdf_set_creator

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_creator -- Sets the creator field in the pdf document

Description

void **cpdf_set_creator** (string creator)

The **cpdf_set_creator()** function sets the creator of a pdf document.

See also **cpdf_set_subject()**, **cpdf_set_title()**, **cpdf_set_keywords()**.

cpdf_set_current_page

(PHP 3>= 3.0.9, PHP 4)

cpdf_set_current_page -- Sets current page

Description

void **cpdf_set_current_page** (int pdf document, int page number)

The **cpdf_set_current_page()** function set the page on which all operations are performed. One can switch between pages until a page is finished with **cpdf_finalize_page()**.

See also **cpdf_finalize_page()**.

cpdf_set_font_directories

(PHP 4 >= 4.0.6)

cpdf_set_font_directories -- Sets directories to search when using external fonts

Description

void **cpdf_set_font_directories** (int pdfdoc, string pfmdir, string pfbdir)

Warning
This function is currently not documented: only the argument list is available.

cpdf_set_font_map_file

(PHP 4 >= 4.0.6)

cpdf_set_font_map_file -- Sets fontname to filename translation map when using external fonts

Description

void **cpdf_set_font_map_file** (int pdfdoc, string filename)

Warning
This function is currently not documented: only the argument list is available.

cpdf_set_font

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_font -- Select the current font face and size

Description

void **cpdf_set_font** (int pdf document, string font name, float size, string encoding)

The **cpdf_set_font()** function sets the current font face, font size and encoding. Currently only the standard postscript fonts are supported.

The **cpdf_set_font()** function sets the current font face, font size and encoding. Currently only the standard postscript fonts are supported.

The last parameter encoding can take the following values: "MacRomanEncoding", "MacExpertEncoding", "WinAnsiEncoding", and "**NULL**". "**NULL**" stands for the font's built-in encoding.

See the ClibPDF Manual for more information, especially how to support asian fonts.

cpdf_set_horiz_scaling

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_horiz_scaling -- Sets horizontal scaling of text

Description

void **cpdf_set_horiz_scaling** (int pdf document, float scale)

The **cpdf_set_horiz_scaling()** function sets the horizontal scaling to scale percent.

cpdf_set_keywords

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_keywords -- Sets the keywords field of the pdf document

Description

void **cpdf_set_keywords** (string keywords)

The **cpdf_set_keywords()** function sets the keywords of a pdf document.

See also **cpdf_set_title()**, **cpdf_set_creator()**, **cpdf_set_subject()**.

cpdf_set_leading

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_leading -- Sets distance between text lines

Description

void **cpdf_set_leading** (int pdf document, float distance)

The **cpdf_set_leading()** function sets the distance between text lines. This will be used if text is output by **cpdf_continue_text()**.

See also **cpdf_continue_text()**.

cpdf_set_page_animation

(PHP 3>= 3.0.9, PHP 4)

cpdf_set_page_animation -- Sets duration between pages

Description

void **cpdf_set_page_animation** (int pdf document, int transition, float duration)

The **cpdf_set_page_animation()** function set the transition between following pages.

The value of transition can be

0 for none,
1 for two lines sweeping across the screen reveal the page,
2 for multiple lines sweeping across the screen reveal the page,
3 for a box reveals the page,
4 for a single line sweeping across the screen reveals the page,
5 for the old page dissolves to reveal the page,
6 for the dissolve effect moves from one screen edge to another,
7 for the old page is simply replaced by the new page (default)

The value of duration is the number of seconds between page flipping.

cpdf_set_subject

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_subject -- Sets the subject field of the pdf document

Description

void **cpdf_set_subject** (string subject)

The **cpdf_set_subject()** function sets the subject of a pdf document.

See also **cpdf_set_title()**, **cpdf_set_creator()**, **cpdf_set_keywords()**.

The **cpdf_set_subject()** function sets the subject of a pdf document.

See also **cpdf_set_title()**, **cpdf_set_creator()**, **cpdf_set_keywords()**.

cpdf_set_text_matrix

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_text_matrix -- Sets the text matrix

Description

void **cpdf_set_text_matrix** (int pdf document, array matrix)

The **cpdf_set_text_matrix()** function sets a matrix which describes a transformation applied on the current text font.

cpdf_set_text_pos

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_text_pos -- Sets text position

Description

void **cpdf_set_text_pos** (int pdf document, float x-coor, float y-coor [, int mode])

The **cpdf_set_text_pos()** function sets the position of text for the next **cpdf_show()** function call.

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

See also **cpdf_show()**, **cpdf_text()**.

cpdf_set_text_rendering

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_text_rendering -- Determines how text is rendered

Description

void **cpdf_set_text_rendering** (int pdf document, int mode)

The **cpdf_set_text_rendering()** function determines how text is rendered.

The possible values for mode are 0=fill text, 1=stroke text, 2=fill and stroke text, 3=invisible, 4=fill text and add it to clipping path, 5=stroke text and add it to clipping path, 6=fill and stroke text and add it to clipping path, 7=add it to clipping path.

cpdf_set_text_rise

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_text_rise -- Sets the text rise

Description

void **cpdf_set_text_rise** (int pdf document, float value)

The **cpdf_set_text_rise()** function sets the text rising to value units.

cpdf_set_title

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_title -- Sets the title field of the pdf document

Description

void **cpdf_set_title** (string title)

The **cpdf_set_title()** function sets the title of a pdf document.

See also **cpdf_set_subject()**, **cpdf_set_creator()**, **cpdf_set_keywords()**.

cpdf_set_viewer_preferences

(PHP 3>= 3.0.9, PHP 4)

cpdf_set_viewer_preferences -- How to show the document in the viewer

Description

void **cpdf_set_viewer_preferences** (int pdfdoc, array preferences)

Warning
This function is currently not documented: only the argument list is available.

This function is currently not documented: only the argument list is available.

cpdf_set_word_spacing

(PHP 3>= 3.0.8, PHP 4)

cpdf_set_word_spacing -- Sets spacing between words

Description

void **cpdf_set_word_spacing** (int pdf document, float space)

The **cpdf_set_word_spacing()** function sets the spacing between words.

See also **cpdf_set_char_spacing()**, **cpdf_set_leading()**.

cpdf_setdash

(PHP 3>= 3.0.8, PHP 4)

cpdf_setdash -- Sets dash pattern

Description

void **cpdf_setdash** (int pdf document, float white, float black)

The **cpdf_setdash()** function set the dash pattern white white units and black black units. If both are 0 a solid line is set.

cpdf_setflat

(PHP 3>= 3.0.8, PHP 4)

cpdf_setflat -- Sets flatness

Description

void **cpdf_setflat** (int pdf document, float value)

The **cpdf_setflat()** function set the flatness to a value between 0 and 100.

cpdf_setgray_fill

(PHP 3>= 3.0.8, PHP 4)

cpdf_setgray_fill -- Sets filling color to gray value

Description

void **cpdf_setgray_fill** (int pdf document, float value)

The **cpdf_setgray_fill()** function sets the current gray value to fill a path.

See also **cpdf_setrgbcolor_fill()**.

cpdf_setgray_stroke

(PHP 3>= 3.0.8, PHP 4)

cpdf_setgray_stroke -- Sets drawing color to gray value

Description

void **cpdf_setgray_stroke** (int pdf document, float gray value)

The **cpdf_setgray_stroke()** function sets the current drawing color to the given gray value.

See also **cpdf_setrgbcolor_stroke()**.

cpdf_setgray

(PHP 3>= 3.0.8, PHP 4)

cpdf_setgray -- Sets drawing and filling color to gray value

Description

void **cpdf_setgray** (int pdf document, float gray value)

The **cpdf_setgray()** function sets the current drawing and filling color to the given gray value.

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor_fill()**.

cpdf_setlinecap

(PHP 3>= 3.0.8, PHP 4)

cpdf_setlinecap -- Sets linecap parameter

cpdf_setlinecap -- Sets linecap parameter

Description

void **cpdf_setlinecap** (int pdf document, int value)

The **cpdf_setlinecap()** function set the linecap parameter between a value of 0 and 2. 0 = butt end, 1 = round, 2 = projecting square.

cpdf_setlinejoin

(PHP 3>= 3.0.8, PHP 4)

cpdf_setlinejoin -- Sets linejoin parameter

Description

void **cpdf_setlinejoin** (int pdf document, long value)

The **cpdf_setlinejoin()** function set the linejoin parameter between a value of 0 and 2. 0 = miter, 1 = round, 2 = bevel.

cpdf_setlinewidth

(PHP 3>= 3.0.8, PHP 4)

cpdf_setlinewidth -- Sets line width

Description

void **cpdf_setlinewidth** (int pdf document, float width)

The **cpdf_setlinewidth()** function set the line width to width.

cpdf_setmiterlimit

(PHP 3>= 3.0.8, PHP 4)

cpdf_setmiterlimit -- Sets miter limit

Description

void **cpdf_setmiterlimit** (int pdf document, float value)

The **cpdf_setmiterlimit()** function set the miter limit to a value greater or equal than 1.

cpdf_setrgbcolor_fill

(PHP 3>= 3.0.8, PHP 4)

cpdf_setrgbcolor_fill -- Sets filling color to rgb color value

Description

void **cpdf_setrgbcolor_fill** (int pdf document, float red value, float green value, float blue value)

The **cpdf_setrgbcolor_fill()** function sets the current rgb color value to fill a path.

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor()**.

cpdf_setrgbcolor_stroke

(PHP 3>= 3.0.8, PHP 4)

cpdf_setrgbcolor_stroke -- Sets drawing color to rgb color value

Description

void **cpdf_setrgbcolor_stroke** (int pdf document, float red value, float green value, float blue value)

The **cpdf_setrgbcolor_stroke()** function sets the current drawing color to the given rgb color value.

See also **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_setrgbcolor

(PHP 3>= 3.0.8, PHP 4)

cpdf_setrgbcolor -- Sets drawing and filling color to rgb color value

Description

void **cpdf_setrgbcolor** (int pdf document, float red value, float green value, float blue value)

The **cpdf_setrgbcolor()** function sets the current drawing and filling color to the given rgb color value.

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor_fill()**.

cpdf_show_xy

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor_fill()**.

cpdf_show_xy

(PHP 3>= 3.0.8, PHP 4)

cpdf_show_xy -- Output text at position

Description

void **cpdf_show_xy** (int pdf document, string text, float x-coor, float y-coor [, int mode])

The **cpdf_show_xy()** function outputs the string text at position with coordinates (x-coor, y-coor).

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

Note: The function **cpdf_show_xy()** is identical to **cpdf_text()** without the optional parameters.

See also **cpdf_text()**.

cpdf_show

(PHP 3>= 3.0.8, PHP 4)

cpdf_show -- Output text at current position

Description

void **cpdf_show** (int pdf document, string text)

The **cpdf_show()** function outputs the string in text at the current position.

See also **cpdf_text()**, **cpdf_begin_text()**, **cpdf_end_text()**.

cpdf_stringwidth

(PHP 3>= 3.0.8, PHP 4)

cpdf_stringwidth -- Returns width of text in current font

Description

float **cpdf_stringwidth** (int pdf document, string text)

The **cpdf_stringwidth()** function returns the width of the string in text. It requires a font to be set before.

See also **cpdf_set_font()**.

cpdf_stroke

(PHP 3>= 3.0.8, PHP 4)

cpdf_stroke -- Draw line along path

Description

void **cpdf_stroke** (int pdf document)

The **cpdf_stroke()** function draws a line along current path.

See also **cpdf_closepath()**, **cpdf_closepath_stroke()**.

cpdf_text

(PHP 3>= 3.0.8, PHP 4)

cpdf_text -- Output text with parameters

Description

void **cpdf_text** (int pdf document, string text, float x-coor, float y-coor [, int mode [, float orientation [, int alignmode]]])

The **cpdf_text()** function outputs the string text at position with coordinates (x-coor, y-coor).

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit. The optional parameter orientation is the rotation of the text in degree. The optional parameter alignmode determines how the text is aligned.

See the ClibPDF documentation for possible values.

See also **cpdf_show_xy()**.

cpdf_translate

(PHP 3>= 3.0.8, PHP 4)

cpdf_translate -- Sets origin of coordinate system

Description

Description

void **cpdf_translate** (int pdf document, float x-coor, float y-coor [, int mode])

The **cpdf_translate()** function set the origin of coordinate system to the point (x-coor, y-coor).

The optional parameter mode determines the unit length. If it's 0 or omitted the default unit as specified for the page is used. Otherwise the coordinates are measured in postscript points disregarding the current unit.

XI. Crack functions

Introduction

These functions allow you to use the CrackLib library to test the 'strength' of a password. The 'strength' of a password is tested by that checks length, use of upper and lower case and checked against the specified CrackLib dictionary. CrackLib will also give helpful diagnostic messages that will help 'strengthen' the password.

Requirements

More information regarding CrackLib along with the library can be found at http://www.users.dircon.co.uk/~crypto/.

Installation

In order to use these functions, you must compile PHP with Crack support by using the --with-crack[=DIR] option.

Runtime Configuration

The behaviour of these functions is affected by settings in php.ini.

Table 1. Crack configuration options

Name	Default	Changeable
crack.default_dictionary	NULL	PHP_INI_SYSTEM

For further details and definition of the PHP_INI_* constants see **ini_set()**.

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

Examples

This example shows how to open a CrackLib dictionary, test a given password, retrieve any diagnostic messages, and close the dictionary.

Example 1. CrackLib example

```
<?php
// Open CrackLib Dictionary
$dictionary = crack_opendict('/usr/local/lib/pw_dict')
    or die('Unable to open CrackLib dictionary');

// Perform password check
$check = crack_check($dictionary, 'gx9A2s0x');

// Retrieve messages
$diag = crack_getlastmessage();
echo $diag; // 'strong password'

// Close dictionary
crack_closedict($dictionary);
?>
```

Note: If **crack_check()** returns **TRUE**, **crack_getlastmessage()** will return 'strong password'.

Table of Contents

crack_check

(PHP 4 >= 4.0.5)

crack_check -- Performs an obscure check with the given password

Description

bool **crack_check** ([resource dictionary, string password])

Returns **TRUE** if password is strong, or **FALSE** otherwise.

Returns **TRUE** if password is strong, or **FALSE** otherwise.

Warning
This function is EXPERIMENTAL. The behaviour of this function, the name of this function, and anything else documented about this function may change without notice in a future release of PHP. Use this function at your own risk.

crack_check() performs an obscure check with the given password on the specified dictionary . If dictionary is not specified, the last opened dictionary is used.

crack_closedict

(PHP 4 >= 4.0.5)

crack_closedict -- Closes an open CrackLib dictionary

Description

bool **crack_closedict** ([resource dictionary])

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

Warning
This function is EXPERIMENTAL. The behaviour of this function, the name of this function, and anything else documented about this function may change without notice in a future release of PHP. Use this function at your own risk.

crack_closedict() closes the specified dictionary identifier. If dictionary is not specified, the current dictionary is closed.

crack_getlastmessage

(PHP 4 >= 4.0.5)

crack_getlastmessage -- Returns the message from the last obscure check

Description

string **crack_getlastmessage** (void)

Warning
This function is EXPERIMENTAL. The behaviour of this function, the name of this function, and anything else documented about this function may change without notice in a future release of PHP. Use this function at your own risk.

crack_getlastmessage() returns the message from the last obscure check.

crack_opendict

(PHP 4 >= 4.0.5)

crack_opendict -- Opens a new CrackLib dictionary

Description

resource **crack_opendict** (string dictionary)

Returns a dictionary resource identifier on success, or **FALSE** on failure.

Warning
This function is EXPERIMENTAL. The behaviour of this function, the name of this function, and anything else documented about this function may change without notice in a future release of PHP. Use this function at your own risk.

crack_opendict() opens the specified CrackLib dictionary for use with **crack_check()**.

> **Note:** Only one dictionary may be open at a time.

See also: **crack_check()**, and **crack_closedict()**.

XII. CURL, Client URL Library Functions

Introduction

PHP supports libcurl, a library created by Daniel Stenberg, that allows you to connect and communicate to many different types of servers with many different types of protocols. libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols. libcurl also supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading (this can also be done with PHP's ftp extension), HTTP form based upload, proxies, cookies, and user+password authentication.

These functions have been added in PHP 4.0.2.

Requirements

In order to use the CURL functions you need to install the CURL package. PHP requires that you use CURL 7.0.2-beta or higher. PHP will not work with any version of CURL below version 7.0.2-beta. From PHP version 4.2.3 you will atleast need CURL version 7.9.0 or higher.

PHP will not work with any version of CURL below version 7.0.2-beta. From PHP version 4.2.3 you will atleast need CURL version 7.9.0 or higher.

Installation

To use PHP's CURL support you must also compile PHP --with-curl[=DIR] where DIR is the location of the directory containing the lib and include directories. In the "include" directory there should be a folder named "curl" which should contain the easy.h and curl.h files. There should be a file named libcurl.a located in the "lib" directory. Beginning with PHP 4.3.0 you can configure PHP to use CURL for url streams --with-curlwrappers.

> **Note to Win32 Users:** In order to enable this module on a Windows environment, you must copy libeay32.dll and ssleay32.dll from the DLL folder of the PHP/Win32 binary package to the SYSTEM32 folder of your windows machine. (Ex: C:\WINNT\SYSTEM32 or C:\WINDOWS\SYSTEM32)

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.

CURLOPT_PORT (integer)

CURLOPT_FILE (integer)

CURLOPT_INFILE (integer)

CURLOPT_INFILESIZE (integer)

CURLOPT_URL (integer)

CURLOPT_PROXY (integer)

CURLOPT_VERBOSE (integer)

CURLOPT_HEADER (integer)

CURLOPT_HTTPHEADER (integer)

CURLOPT_NOPROGRESS (integer)

CURLOPT_NOBODY (integer)

CURLOPT_FAILONERROR (integer)

CURLOPT_UPLOAD (integer)

CURLOPT_POST (integer)

CURLOPT_FTPLISTONLY (integer)

CURLOPT_FTPAPPEND (integer)

CURLOPT_NETRC (integer)

CURLOPT_FOLLOWLOCATION (integer)

CURLOPT_FTPASCII (integer)

CURLOPT_PUT (integer)

CURLOPT_MUTE (integer)

CURLOPT_USERPWD (integer)

CURLOPT_PROXYUSERPWD (integer)

CURLOPT_RANGE (integer)

CURLOPT_TIMEOUT (integer)

CURLOPT_POSTFIELDS (integer)

CURLOPT_REFERER (integer)

CURLOPT_USERAGENT (integer)

CURLOPT_FTPPORT (integer)

CURLOPT_LOW_SPEED_LIMIT (integer)

CURLOPT_LOW_SPEED_TIME (integer)

CURLOPT_RESUME_FROM (integer)

CURLOPT_COOKIE (integer)

CURLOPT_SSLCERT (integer)

CURLOPT_SSLCERTPASSWD (integer)

CURLOPT_WRITEHEADER (integer)

CURLOPT_SSL_VERIFYHOST (integer)

CURLOPT_COOKIEFILE (integer)

CURLOPT_SSLVERSION (integer)

CURLOPT_TIMECONDITION (integer)

CURLOPT_TIMEVALUE (integer)

CURLOPT_CUSTOMREQUEST (integer)

CURLOPT_STDERR (integer)

CURLOPT_STDERR (integer)

CURLOPT_TRANSFERTEXT (integer)

CURLOPT_RETURNTRANSFER (integer)

CURLOPT_QUOTE (integer)

CURLOPT_POSTQUOTE (integer)

CURLOPT_INTERFACE (integer)

CURLOPT_KRB4LEVEL (integer)

CURLOPT_HTTPPROXYTUNNEL (integer)

CURLOPT_FILETIME (integer)

CURLOPT_WRITEFUNCTION (integer)

CURLOPT_READFUNCTION (integer)

CURLOPT_PASSWDFUNCTION (integer)

CURLOPT_HEADERFUNCTION (integer)

CURLOPT_MAXREDIRS (integer)

CURLOPT_MAXCONNECTS (integer)

CURLOPT_CLOSEPOLICY (integer)

CURLOPT_FRESH_CONNECT (integer)

CURLOPT_FORBID_REUSE (integer)

CURLOPT_RANDOM_FILE (integer)

CURLOPT_EGDSOCKET (integer)

CURLOPT_CONNECTTIMEOUT (integer)

CURLOPT_SSL_VERIFYPEER (integer)

CURLOPT_CAINFO (integer)

CURLOPT_COOKIEJAR (integer)

CURLOPT_SSL_CIPHER_LIST (integer)

CURLOPT_BINARYTRANSFER (integer)

CURLCLOSEPOLICY_LEAST_RECENTLY_USED (integer)

CURLCLOSEPOLICY_LEAST_TRAFFIC (integer)

CURLCLOSEPOLICY_SLOWEST (integer)

CURLCLOSEPOLICY_CALLBACK (integer)

CURLCLOSEPOLICY_OLDEST (integer)

CURLINFO_EFFECTIVE_URL (integer)

CURLINFO_HTTP_CODE (integer)

CURLINFO_HEADER_SIZE (integer)

CURLINFO_REQUEST_SIZE (integer)

CURLINFO_TOTAL_TIME (integer)

CURLINFO_NAMELOOKUP_TIME (integer)

CURLINFO_CONNECT_TIME (integer)

CURLINFO_PRETRANSFER_TIME (integer)

CURLINFO_SIZE_UPLOAD (integer)

CURLINFO_SIZE_DOWNLOAD (integer)

CURLINFO_SPEED_DOWNLOAD (integer)

CURLINFO_SPEED_UPLOAD (integer)

CURLINFO_FILETIME (integer)

CURLINFO_SSL_VERIFYRESULT (integer)

CURLINFO_CONTENT_LENGTH_DOWNLOAD (integer)

CURLINFO_CONTENT_LENGTH_UPLOAD (integer)

CURLE_OK (integer)

CURLE_UNSUPPORTED_PROTOCOL (integer)

CURLE_FAILED_INIT (integer)

CURLE_URL_MALFORMAT (integer)

CURLE_URL_MALFORMAT_USER (integer)

CURLE_COULDNT_RESOLVE_PROXY (integer)

CURLE_COULDNT_RESOLVE_HOST (integer)

CURLE_COULDNT_CONNECT (integer)

CURLE_FTP_WEIRD_SERVER_REPLY (integer)

CURLE_FTP_WEIRD_SERVER_REPLY (integer)

CURLE_FTP_ACCESS_DENIED (integer)

CURLE_FTP_USER_PASSWORD_INCORRECT (integer)

CURLE_FTP_WEIRD_PASS_REPLY (integer)

CURLE_FTP_WEIRD_USER_REPLY (integer)

CURLE_FTP_WEIRD_PASV_REPLY (integer)

CURLE_FTP_WEIRD_227_FORMAT (integer)

CURLE_FTP_CANT_GET_HOST (integer)

CURLE_FTP_CANT_RECONNECT (integer)

CURLE_FTP_COULDNT_SET_BINARY (integer)

CURLE_PARTIAL_FILE (integer)

CURLE_FTP_COULDNT_RETR_FILE (integer)

CURLE_FTP_WRITE_ERROR (integer)

CURLE_FTP_QUOTE_ERROR (integer)

CURLE_HTTP_NOT_FOUND (integer)

CURLE_WRITE_ERROR (integer)

CURLE_MALFORMAT_USER (integer)

CURLE_FTP_COULDNT_STOR_FILE (integer)

CURLE_READ_ERROR (integer)

CURLE_OUT_OF_MEMORY (integer)

CURLE_OPERATION_TIMEOUTED (integer)

CURLE_FTP_COULDNT_SET_ASCII (integer)

CURLE_FTP_PORT_FAILED (integer)

CURLE_FTP_COULDNT_USE_REST (integer)

CURLE_FTP_COULDNT_GET_SIZE (integer)

CURLE_HTTP_RANGE_ERROR (integer)

CURLE_HTTP_POST_ERROR (integer)

CURLE_SSL_CONNECT_ERROR (integer)

CURLE_FTP_BAD_DOWNLOAD_RESUME (integer)

CURLE_FILE_COULDNT_READ_FILE (integer)

CURLE_LDAP_CANNOT_BIND (integer)

CURLE_LDAP_SEARCH_FAILED (integer)

CURLE_LIBRARY_NOT_FOUND (integer)

CURLE_FUNCTION_NOT_FOUND (integer)

CURLE_ABORTED_BY_CALLBACK (integer)

CURLE_BAD_FUNCTION_ARGUMENT (integer)

CURLE_BAD_CALLING_ORDER (integer)

CURLE_HTTP_PORT_FAILED (integer)

CURLE_BAD_PASSWORD_ENTERED (integer)

CURLE_TOO_MANY_REDIRECTS (integer)

CURLE_UNKNOWN_TELNET_OPTION (integer)

CURLE_TELNET_OPTION_SYNTAX (integer)

CURLE_OBSOLETE (integer)

CURLE_SSL_PEER_CERTIFICATE (integer)

Examples

Once you've compiled PHP with CURL support, you can begin using the CURL functions. The basic idea behind the CURL functions is that you initialize a CURL session using the **curl_init()**, then you can set all your options for the transfer via the **curl_setopt()**, then you can execute the session with the **curl_exec()** and then you finish off your session using the **curl_close()**. Here is an example that uses the CURL functions to fetch the example.com homepage into a file:

Example 1. Using PHP's CURL module to fetch the example.com homepage

```
<?php

$ch = curl_init ("http://www.example.com/");
$fp = fopen ("example_homepage.txt", "w");

curl_setopt ($ch, CURLOPT_FILE, $fp);
curl_setopt ($ch, CURLOPT_HEADER, 0);
```

```
curl_setopt ($ch, CURLOPT_HEADER, 0);

curl_exec ($ch);
curl_close ($ch);
fclose ($fp);
?>
```

Table of Contents

curl_close

(PHP 4 >= 4.0.2)

curl_close -- Close a CURL session

Description

void **curl_close** (resource ch)

This function closes a CURL session and frees all resources. The CURL handle, ch, is also deleted.

curl_errno

(PHP 4 >= 4.0.3)

curl_errno -- Return an integer containing the last error number

Description

int **curl_errno** (resource ch)

Warning
This function is currently not documented; only the argument list is available.

curl_error

(PHP 4 >= 4.0.3)

curl_error -- Return a string containing the last error for the current session

Description

string **curl_error** (resource ch)

Warning
This function is currently not documented; only the argument list is available.

curl_exec

(PHP 4 >= 4.0.2)

curl_exec -- Perform a CURL session

Description

bool **curl_exec** (resource ch)

This function should be called after you initialize a CURL session and all the options for the session are set. Its purpose is simply to execute the predefined CURL session (given by the ch).

> **Tip:** As with anything that outputs its result directly to the browser, you can use the output-control functions to capture the output of this function, and save it in a **string** (for example).

curl_getinfo

(PHP 4 >= 4.0.4)

curl_getinfo -- Get information regarding a specific transfer

Description

string **curl_getinfo** (resource ch [, int opt])

Returns information about the last transfer, opt may be one of the following:

Returns information about the last transfer, opt may be one of the following:

- "CURLINFO_EFFECTIVE_URL" - Last effective URL
- "CURLINFO_HTTP_CODE" - Last received HTTP code
- "CURLINFO_FILETIME" - Remote time of the retrieved document, if -1 is returned the time of the document is unknown
- "CURLINFO_TOTAL_TIME" - Total transaction time in seconds for last transfer
- "CURLINFO_NAMELOOKUP_TIME" - Time in seconds until name resolving was complete
- "CURLINFO_CONNECT_TIME" - Time in seconds it took to establish the connection
- "CURLINFO_PRETRANSFER_TIME" - Time in seconds from start until just before file transfer begins
- "CURLINFO_STARTTRANSFER_TIME" - Time in seconds until the first byte is about to be transfered
- "CURLINFO_REDIRECT_TIME" - Time in seconds of all redirection steps before final transaction was started
- "CURLINFO_SIZE_UPLOAD" - Total number of bytes uploaded
- "CURLINFO_SIZE_DOWNLOAD" - Total number of bytes downloaded
- "CURLINFO_SPEED_DOWNLOAD" - Average download speed
- "CURLINFO_SPEED_UPLOAD" - Average upload speed
- "CURLINFO_HEADER_SIZE" - Total size of all headers received
- "CURLINFO_REQUEST_SIZE" - Total size of issued requests, currently only for HTTP requests
- "CURLINFO_SSL_VERIFYRESULT" - Result of SSL certification verification requested by setting CURLOPT_SSL_VERIFYPEER
- "CURLINFO_CONTENT_LENGTH_DOWNLOAD" - content-length of download, read from Content-Length: field
- "CURLINFO_CONTENT_LENGTH_UPLOAD" - Specified size of upload
- "CURLINFO_CONTENT_TYPE" - Content-type of downloaded object, NULL indicates server did not send valid Content-Type: header

If called without the optional parameter opt an assoctive array is returned with the following array elements which correspond to opt options:

- "url"
- "content_type"
- "http_encode"
- "header_size"
- "request_size"
- "filetime"
- "ssl_verify_result"
- "redirect_count"
- "total_time"
- "namelookup_time"
- "connect_time"
- "pretransfer_time"
- "size_upload"
- "size_download"
- "speed_download"
- "speed_upload"
- "download_content_length"
- "upload_content_length"
- "starttransfer_time"
- "redirect_time"

curl_init

(PHP 4 >= 4.0.2)

curl_init -- Initialize a CURL session

Description

resource **curl_init** ([string url])

The **curl_init()** will initialize a new session and return a CURL handle for use with the **curl_setopt()**, **curl_exec()**, and **curl_close()** functions. If the optional url parameter is supplied then the CURLOPT_URL option will be set to the value of the parameter. You can manually set this using the **curl_setopt()** function.

Example 1. Initializing a new CURL session and fetching a webpage

```
<?php
$ch = curl_init();

curl_setopt ($ch, CURLOPT_URL, "http://www.example.com/");
curl_setopt ($ch, CURLOPT_HEADER, 0);

curl_exec ($ch);

curl_close ($ch);
?>
```

See also: **curl_close()**, **curl_setopt()**

curl_setopt

(PHP 4 >= 4.0.2)

curl_setopt -- Set an option for a CURL transfer

Description

bool **curl_setopt** (resource ch, string option, mixed value)

The **curl_setopt()** function will set options for a CURL session identified by the ch parameter. The option parameter is the option you want to set, and the value is the value of the option given by the option.

The value should be a long for the following options (specified in the option parameter):

The value should be a long for the following options (specified in the option parameter):

- CURLOPT_INFILESIZE: When you are uploading a file to a remote site, this option should be used to tell PHP what the expected size of the infile will be.
- CURLOPT_VERBOSE: Set this option to a non-zero value if you want CURL to report everything that is happening.
- CURLOPT_HEADER: Set this option to a non-zero value if you want the header to be included in the output.
- CURLOPT_NOPROGRESS: Set this option to a non-zero value if you don't want PHP to display a progress meter for CURL transfers.

  Note: PHP automatically sets this option to a non-zero parameter, this should only be changed for debugging purposes.
- CURLOPT_NOBODY: Set this option to a non-zero value if you don't want the body included with the output.
- CURLOPT_FAILONERROR: Set this option to a non-zero value if you want PHP to fail silently if the HTTP code returned is greater than 300. The default behavior is to return the page normally, ignoring the code.
- CURLOPT_UPLOAD: Set this option to a non-zero value if you want PHP to prepare for an upload.
- CURLOPT_POST: Set this option to a non-zero value if you want PHP to do a regular HTTP POST. This POST is a normal application/x-www-form-urlencoded kind, most commonly used by HTML forms.
- CURLOPT_FTPLISTONLY: Set this option to a non-zero value and PHP will just list the names of an FTP directory.
- CURLOPT_FTPAPPEND: Set this option to a non-zero value and PHP will append to the remote file instead of overwriting it.
- CURLOPT_NETRC: Set this option to a non-zero value and PHP will scan your ~./netrc file to find your username and password for the remote site that you're establishing a connection with.
- CURLOPT_FOLLOWLOCATION: Set this option to a non-zero value to follow any "Location: " header that the server sends as a part of the HTTP header (note this is recursive, PHP will follow as many "Location: " headers that it is sent.)
- CURLOPT_PUT: Set this option to a non-zero value to HTTP PUT a file. The file to PUT must be set with the CURLOPT_INFILE and CURLOPT_INFILESIZE.
- CURLOPT_MUTE: Set this option to a non-zero value and PHP will be completely silent with regards to the CURL functions.
- CURLOPT_TIMEOUT: Pass a long as a parameter that contains the maximum time, in seconds, that you'll allow the CURL functions to take.
- CURLOPT_LOW_SPEED_LIMIT: Pass a long as a parameter that contains the transfer speed in bytes per second that the transfer should be below during CURLOPT_LOW_SPEED_TIME seconds for PHP to consider it too slow and abort.
- CURLOPT_LOW_SPEED_TIME: Pass a long as a parameter that contains the time in seconds that the transfer should be below the CURLOPT_LOW_SPEED_LIMIT for PHP to consider it too slow and abort.
- CURLOPT_RESUME_FROM: Pass a long as a parameter that contains the offset, in bytes, that you want the transfer to start from.
- CURLOPT_SSLVERSION: Pass a long as a parameter that contains the SSL version (2 or 3) to use. By default PHP will try and determine this by itself, although, in some cases you must set this manually.
- CURLOPT_SSL_VERIFYHOST: Pass a long if CURL should verify the Common name of the peer certificate in the SSL handshake. A value of 1 denotes that we should check for the existence of the common name, a value of 2 denotes that we should make sure it matches the provided hostname.
- CURLOPT_TIMECONDITION: Pass a long as a parameter that defines how the CURLOPT_TIMEVALUE is treated. You can set this parameter to TIMECOND_IFMODSINCE or TIMECOND_ISUNMODSINCE. This is a HTTP-only feature.
- CURLOPT_TIMEVALUE: Pass a long as a parameter that is the time in seconds since January 1st, 1970. The time will be used as specified by the CURLOPT_TIMEVALUE option, or by default the TIMECOND_IFMODSINCE will be used.
- CURLOPT_RETURNTRANSFER: Pass a non-zero value if you want CURL to directly return the transfer instead of printing it out directly.

The value parameter should be a string for the following values of the option parameter:

- CURLOPT_URL: This is the URL that you want PHP to fetch. You can also set this option when initializing a session with the **curl_init()** function.
- CURLOPT_USERPWD: Pass a string formatted in the [username]:[password] manner, for PHP to use for the connection.
- CURLOPT_PROXYUSERPWD: Pass a string formatted in the [username]:[password] format for connection to the HTTP proxy.
- CURLOPT_RANGE: Pass the specified range you want. It should be in the "X-Y" format, where X or Y may be left out. The HTTP transfers also support several intervals, separated with commas as in X-Y,N-M.
- CURLOPT_POSTFIELDS: Pass a string containing the full data to post in an HTTP "POST" operation.
- CURLOPT_REFERER: Pass a string containing the "referer" header to be used in an HTTP request.
- CURLOPT_USERAGENT: Pass a string containing the "user-agent" header to be used in an HTTP request.
- CURLOPT_FTPPORT: Pass a string containing the value which will be used to get the IP address to use for the ftp "POST" instruction. The POST instruction tells the remote server to connect to our specified IP address. The string may be a plain IP address, a hostname, a network interface name (under UNIX), or just a plain '-' to use the systems default IP address.
- CURLOPT_COOKIE: Pass a string containing the content of the cookie to be set in the HTTP header.
- CURLOPT_SSLCERT: Pass a string containing the filename of PEM formatted certificate.
- CURLOPT_SSLCERTPASSWD: Pass a string containing the password required to use the CURLOPT_SSLCERT certificate.
- CURLOPT_COOKIEFILE: Pass a string containing the name of the file containing the cookie data. The cookie file can be in Netscape format, or just plain HTTP-style headers dumped into a file.
- CURLOPT_CUSTOMREQUEST: Pass a string to be used instead of GET or HEAD when doing an HTTP request. This is useful for doing DELETE or other, more obscure, HTTP requests. Valid values are things like GET, POST, and so on; i.e. do not enter a whole HTTP request line here. For instance, entering 'GET /index.html HTTP/1.0\r\n\r\n' would be incorrect.

  Note: Don't do this without making sure your server supports the command first.
- CURLOPT_PROXY: Give the name of the HTTP proxy to tunnel requests through.
- CURLOPT_INTERFACE: Pass the name of the outgoing network interface to use. This can be an interface name, an IP address or a host name.
- CURLOPT_KRB4LEVEL: Pass the KRB4 (Kerberos 4) security level. Anyone of the following strings (in order from least powerful, to most powerful): 'clear', 'safe', 'confidential', 'private'. If the string does not match one of these, then 'private' is used. If you set this to NULL, this disables KRB4 security. KRB4 security only works with FTP transactions currently.
- CURLOPT_HTTPHEADER: Pass an array of HTTP header fields to set.
- CURLOPT_QUOTE: Pass an array of FTP commands to perform on the server prior to the FTP request.
- CURLOPT_POSTQUOTE: Pass an array of FTP commands to execute on the server, after the FTP request has been performed.

The following options expect a file descriptor that is obtained by using the **fopen()** function:

- CURLOPT_FILE: The file where the output of your transfer should be placed, the default is STDOUT.
- CURLOPT_INFILE: The file where the input of your transfer comes from.
- CURLOPT_WRITEHEADER: The file to write the header part of the output into.
- CURLOPT_STDERR: The file to write errors to instead of stderr.

curl_version

curl_version

(PHP 4 >= 4.0.2)

curl_version -- Return the current CURL version

Description

string **curl_version** (void)

The **curl_version()** function returns a string containing the current CURL version.

XIII. Cybercash payment functions

Installation

These functions are only available if the interpreter has been compiled with the --with-cybercash=[DIR].

This extension has been removed from PHP as of PHP 4.3.0, if you have questions as to why then you will find the following CyberCash Faq helpful. In short, CyberCash was bought out by VeriSign and although the CyberCash service continues to exist, VeriSign encourages users to switch. See the above faq for details.

Table of Contents

cybercash_base64_decode

(PHP 4 <= 4.2.3)

cybercash_base64_decode -- base64 decode data for Cybercash

Description

string **cybercash_base64_decode** (string inbuff)

cybercash_base64_encode

(PHP 4 <= 4.2.3)

cybercash_base64_encode -- base64 encode data for Cybercash

Description

string **cybercash_base64_encode** (string inbuff)

cybercash_decr

(PHP 4 <= 4.2.3)

cybercash_decr -- Cybercash decrypt

Description

array **cybercash_decr** (string wmk, string sk, string inbuff)

The function returns an associative array with the elements "errcode" and, if "errcode" is **FALSE**, "outbuff" (string), "outLth" (long) and "macbuff" (string).

cybercash_encr

(PHP 4 <= 4.2.3)

cybercash_encr -- Cybercash encrypt

Description

array **cybercash_encr** (string wmk, string sk, string inbuff)

The function returns an associative array with the elements "errcode" and, if "errcode" is **FALSE**, "outbuff" (string), "outLth" (long) and "macbuff" (string).

XIV. Crédit Mutuel CyberMUT functions

Introduction

This extension allows you to process credit cards transactions using Crédit Mutuel CyberMUT system (http://www.creditmutuel.fr/centre_commercial/vendez_sur_internet.html).

CyberMUT is a popular Web Payment Service in France, provided by the Crédit Mutuel bank. If you are foreign in France, these functions will not be useful for you.

The use of these functions is almost identical to the original SDK functions, except for the parameters of return for **cybermut_creerformulairecm()** and **cybermut_creerreponsecm()**, which are returned directly by functions PHP, whereas they had

The use of these functions is almost identical to the original SDK functions, except for the parameters of return for **cybermut_creerformulairecm()** and **cybermut_creerreponsecm()**, which are returned directly by functions PHP, whereas they had passed in reference in the original functions.

These functions have been added in PHP 4.0.6.

> **Note:** These functions only provide a link to CyberMUT SDK. Be sure to read the CyberMUT Developers Guide for full details of the required parameters.

Installation

These functions are only available if PHP has been compiled with the --with-cybermut[=DIR] option, where DIR is the location of libcm-mac.a and cm-mac.h . You will require the appropriate SDK for your platform, which may be sent to you after your CyberMUT's subscription (contact them via Web, or go to the nearest Crédit Mutuel).

> **Note:** This extension is not available on Windows platforms.

Table of Contents

cybermut_creerformulairecm

(4.0.5 - 4.2.3 only)

cybermut_creerformulairecm -- Generate HTML form of request for payment

Description

string **cybermut_creerformulairecm** (string url_CM, string version, string TPE, string montant, string ref_commande, string texte_libre, string url_retour, string url_retour_ok, string url_retour_err, string langue, string code_societe, string texte_bouton)

cybermut_creerformulairecm() is used to generate the HTML form of request for payment.

Example 1. First step of payment (equiv cgi1.c)

```
<?php
// Directory where the keys are located
putenv("CMKEYDIR=/var/creditmut/cles");

// Version number
$VERSION="1.2";

  $retour =  cybermut_creerformulairecm(
  "https://www.creditmutuel.fr/test/telepaiement/paiement.cgi",
  $VERSION,
  "1234567890",
  "300FRF",
  $REFERENCE,
  $TEXTE_LIBRE,
  $URL_RETOUR,
  $URL_RETOUR_OK,
  $URL_RETOUR_ERR,
  "francais",
  "company",
  "Paiement par carte bancaire");

  echo $retour;
?>
```

See also **cybermut_testmac()** and **cybermut_creerreponsecm()**.

cybermut_creerreponsecm

(4.0.5 - 4.2.3 only)

cybermut_creerreponsecm -- Generate the acknowledgement of delivery of the confirmation of payment

Description

string **cybermut_creerreponsecm** (string phrase)

cybermut_creerreponsecm() returns a string containing delivery acknowledgement message.

The parameter is "OK" if the message of confirmation of the payment was correctly identified by **cybermut_testmac()**. Any other chain is regarded as an error message.

See also **cybermut_creerformulairecm()** and **cybermut_testmac()**.

cybermut_testmac

(4.0.5 - 4.2.3 only)

cybermut_testmac -- Make sure that there no was data diddling contained in the received message of confirmation

Description

bool **cybermut_testmac** (string code_MAC, string version, string TPE, string cdate, string montant, string ref_commande, string texte_libre, string code-retour)

bool **cybermut_testmac** (string code_MAC, string version, string TPE, string cdate, string montant, string ref_commande, string texte_libre, string code-retour)

cybermut_testmac() is used to make sure that there was not data diddling contained in the received message of confirmation. Pay attention to parameters code-retour and texte-libre, which cannot be evaluated as is, because of the dash. You must retrieve them by using:

```
<?php
  $code_retour = $_GET["code-retour"];
  $texte_libre = $_GET["texte-libre"];
?>
```

Example 1. Last step of payment (equiv cgi2.c)

```
<?php
// Make sure that Enable Track Vars is ON.
// Directory where are located the keys
putenv("CMKEYDIR=/var/creditmut/cles");

// Version number
$VERSION="1.2";

$texte_libre = $_GET["texte-libre"];
$code_retour = $_GET["code-retour"];

$mac_ok = cybermut_testmac($MAC,$VERSION,$TPE,$date,$montant,$reference,$texte_libre,$code_retour);

if ($mac_ok) {

  //
  // insert data processing here
  //
  //

  $result=cybermut_creerreponsecm("OK");
} else {
  $result=cybermut_creerreponsecm("Document Falsifie");
}

?>
```

See also **cybermut_creerformulairecm()** and **cybermut_creerreponsecm()**.

XV. Cyrus IMAP administration functions

Introduction

Warning
This function is currently not documented; only the argument list is available.

Note: This extension is not available on Windows platforms.

Installation

To enable Cyrus IMAP support and to use these functions you have to compile PHP with the --with-cyrus option.

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.

CYRUS_CONN_NONSYNCLITERAL (integer)

CYRUS_CONN_INITIALRESPONSE (integer)

CYRUS_CALLBACK_NUMBERED (integer)

CYRUS_CALLBACK_NOLITERAL (integer)

Table of Contents

cyrus_authenticate

(PHP 4 >= 4.1.0)

cyrus_authenticate -- Authenticate against a Cyrus IMAP server

Description

bool **cyrus_authenticate** (resource connection [, string mechlist [, string service [, string user [, int minssf [, int maxssf]]]]])

Warning
This function is currently not documented: only the argument list is available.

cyrus_bind

(PHP 4 >= 4.1.0)

cyrus_bind -- Bind callbacks to a Cyrus IMAP connection

Description

bool **cyrus_bind** (resource connection, array callbacks)

Warning
This function is currently not documented: only the argument list is available.

cyrus_close

(PHP 4 >= 4.1.0)

cyrus_close -- Close connection to a Cyrus IMAP server

Description

bool **cyrus_close** (resource connection)

Warning
This function is currently not documented: only the argument list is available.

cyrus_connect

(PHP 4 >= 4.1.0)

cyrus_connect -- Connect to a Cyrus IMAP server

Description

resource **cyrus_connect** ([string host [, string port [, int flags]]])

Warning
This function is currently not documented: only the argument list is available.

cyrus_query

(PHP 4 >= 4.1.0)

cyrus_query -- Send a query to a Cyrus IMAP server

Description

bool **cyrus_query** (resource connection, string query)

Warning
This function is currently not documented: only the argument list is available.

cyrus_unbind

(PHP 4 >= 4.1.0)

cyrus_unbind -- Unbind ...

Description

bool **cyrus_unbind** (resource connection, string trigger_name)

Warning
This function is currently not documented: only the argument list is available.

XVI. Character type functions

XVI. Character type functions

Introduction

The functions provided by this extension check whether a character or string falls into a certain character class according to the current locale (see also **setlocale()**).

When called with an integer argument these functions behave exactly like their C counterparts from ctype.h.

When called with a string argument they will check every character in the string and will only return **TRUE** if every character in the string matches the requested criteria. When called with an empty string the result will always be **TRUE**.

Passing anything else but a string or integer will return **FALSE** immediately.

Requirements

None besides functions from the standard C library which are always available.

Installation

Beginning with PHP 4.2.0 these functions are enabled by default. For older versions you have to configure and compile PHP with --enable-ctype. You can disable ctype support with --disable-ctype.

The windows version of PHP has built in support for this extension. You do not need to load any additional extension in order to use these functions.

> **Note:** Builtin support for ctype is available with PHP 4.3.0.

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.

Table of Contents

ctype_alnum

(PHP 4 >= 4.0.4)

ctype_alnum -- Check for alphanumeric character(s)

Description

bool **ctype_alnum** (string text)

Returns **TRUE** if every character in text is either a letter or a digit, **FALSE** otherwise. In the standard C locale letters are just [A-Za-z]. The function is equivalent to (ctype_alpha($text) || ctype_digit($text)).

See also **ctype_alpha()**, **ctype_digit()**, and **setlocale()**.

ctype_alpha

(PHP 4 >= 4.0.4)

ctype_alpha -- Check for alphabetic character(s)

Description

bool **ctype_alpha** (string text)

Returns **TRUE** if every character in text is a letter from the current locale, **FALSE** otherwise. In the standard C locale letters are just [A-Za-z] and **ctype_alpha()** is equivalent to (ctype_upper($text) || ctype_lower($text)), but other languages have letters that are considered neither upper nor lower case.

See also **ctype_upper()**, **ctype_lower()**, and **setlocale()**.

ctype_cntrl

(PHP 4 >= 4.0.4)

(PHP 4 >= 4.0.4)

ctype_cntrl -- Check for control character(s)

Description

bool **ctype_cntrl** (string text)

Returns **TRUE** if every character in text has a special control function, **FALSE** otherwise. Control characters are e.g. line feed, tab, esc.

ctype_digit

(PHP 4 >= 4.0.4)

ctype_digit -- Check for numeric character(s)

Description

bool **ctype_digit** (string text)

Returns **TRUE** if every character in text is a decimal digit, **FALSE** otherwise.

See also **ctype_alnum()** and **ctype_xdigit()**.

ctype_graph

(PHP 4 >= 4.0.4)

ctype_graph -- Check for any printable character(s) except space

Description

bool **ctype_graph** (string text)

Returns **TRUE** if every character in text is printable and actually creates visible output (no white space), **FALSE** otherwise.

See also **ctype_alnum()**, **ctype_print()**, and **ctype_punct()**.

ctype_lower

(PHP 4 >= 4.0.4)

ctype_lower -- Check for lowercase character(s)

Description

bool **ctype_lower** (string text)

Returns **TRUE** if every character in text is a lowercase letter in the current locale.

See also **ctype_alpha()** and **ctype_upper()**.

ctype_print

(PHP 4 >= 4.0.4)

ctype_print -- Check for printable character(s)

Description

bool **ctype_print** (string text)

Returns **TRUE** if every character in text will actually create output (including blanks). Returns **FALSE** if text contains control characters or characters that do not have any output or control function at all.

See also **ctype_cntrl()**, **ctype_graph()**, and **ctype_punct()**.

ctype_punct

(PHP 4 >= 4.0.4)

ctype_punct -- Check for any printable character which is not whitespace or an alphanumeric character

Description

bool **ctype_punct** (string text)

Returns **TRUE** if every character in text is printable, but neither letter, digit or blank, **FALSE** otherwise.

See also **ctype_cntrl()**, **ctype_graph()**, and **ctype_punct()**.

ctype_space

(PHP 4 >= 4.0.4)

ctype_space -- Check for whitespace character(s)

ctype_space -- Check for whitespace character(s)

Description

bool **ctype_space** (string text)

Returns **TRUE** if every character in text creates some sort of white space, **FALSE** otherwise. Besides the blank character this also includes tab, vertical tab, line feed, carriage return and form feed characters.

ctype_upper

(PHP 4 >= 4.0.4)

ctype_upper -- Check for uppercase character(s)

Description

bool **ctype_upper** (string text)

Returns **TRUE** if every character in text is a uppercase letter in the current locale.

See also **ctype_alpha()** and **ctype_lower()**.

ctype_xdigit

(PHP 4 >= 4.0.4)

ctype_xdigit -- Check for character(s) representing a hexadecimal digit

Description

bool **ctype_xdigit** (string text)

Returns **TRUE** if every character in text is a hexadecimal 'digit', that is a decimal digit or a character from [A-Fa-f] , **FALSE** otherwise.

See also **ctype_digit()**.

XVII. Database (dbm-style) abstraction layer functions

Introduction

These functions build the foundation for accessing Berkeley DB style databases.

This is a general abstraction layer for several file-based databases. As such, functionality is limited to a common subset of features supported by modern databases such as Sleepycat Software's DB2. (This is not to be confused with IBM's DB2 software, which is supported through the ODBC functions.)

Requirements

The behaviour of various aspects depends on the implementation of the underlying database. Functions such as **dba_optimize()** and **dba_sync()** will do what they promise for one database and will do nothing for others. You have to download and install supported dba-Handlers.

Table 1. List of DBA handlers

Handler	Notes
dbm	Dbm is the oldest (original) type of Berkeley DB style databases. You should avoid it, if possible. We do not support the compatibility functions built into DB2 and gdbm, because they are only compatible on the source code level, but cannot handle the original dbm format.
ndbm	Ndbm is a newer type and more flexible than dbm. It still has most of the arbitrary limits of dbm (therefore it is deprecated).
gdbm	Gdbm is the GNU database manager.
db2	DB2 is Sleepycat Software's DB2. It is described as "a programmatic toolkit that provides high-performance built-in database support for both standalone and client/server applications.
db3	DB3 is Sleepycat Software's DB3.
db4	DB4 is Sleepycat Software's DB4. This is available since PHP 5.0.0.
cdb	Cdb is "a fast, reliable, lightweight package for creating and reading constant databases." It is from the author of qmail and can be found here. Since it is constant, we support only reading operations. And since PHP 4.3.0 we support writing (not updating) through the internal cdb library.
cdb_make	Since PHP 4.3.0 we support creation (not updating) of cdb files when the bundeled cdb library is used.
flatfile	This is available since PHP 4.3.0 for compatibility with the deprecated dbm extension only and should be avoided. However you may use this where files were created in this format. That happens when configure could not find any external library.

When invoking the **dba_open()** or **dba_popen()** functions, one of the handler names must be supplied as an argument. The actually available list of handlers is displayed by invoking **phpinfo()** or **dba_handlers()**.

Installation

By using the --enable-dba=shared configuration option you can build a dynamic loadable modul to enable PHP for basic support of dbm-style databases. You also have to add support for at least one of the following handlers by specifying the --with-XXXX configure switch to your PHP configure line.

Table 2. Supported DBA handlers

Table 2. Supported DBA handlers

Handler	Configure Switch
dbm	To enable support for dbm add --with-dbm[=DIR].
ndbm	To enable support for ndbm add --with-ndbm[=DIR].
gdbm	To enable support for gdbm add --with-gdbm[=DIR].
db2	To enable support for db2 add --with-db2[=DIR]. **Note:** db2 conflicts with db3 and db4.
db3	To enable support for db3 add --with-db3[=DIR]. **Note:** db3 conflicts with db2 and db4.
db4	To enable support for db4 add --with-db4[=DIR]. **Note:** db4 conflicts with db2 and db3. **Note:** This was added in PHP 5.0.0. In earlier version you need to use --with-db3=DIR with DIR being the path to db4 librarie. It is not possible to use db versions starting from 4.1 with PHP prior to version 4.3.0.
cdb	To enable support for cdb add --with-cdb[=DIR]. **Note:** Since PHP 4.3.0 you can omit DIR to use the bundeled cdb library that adds the cdb_make handler which allows creation of cdb files and allows to access cdb files on the network using php's streams.
flatfile	To enable support for flatfile add --with-flatfile. **Note:** This was added in PHP 4.3.0 to add compatibility with deprecated dbm extension. Uee this handler only when you cannot install one of the libraries required by the other handlers and when you cannot use bundeled cdb handler.

Note: Up to PHP 4.3.0 you are able to add both db2 and db3 handler but only one of them can be used internally. That means that you cannot have both file formats. Starting with PHP 5.0.0 there is a configuration check avoid such missconfigurations.

Runtime Configuration

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

Resource Types

The functions **dba_open()** and **dba_popen()** return a handle to the specified database file to access which is used by all other dba-function calls.

Predefined Constants

This extension has no constants defined.

Examples

Example 1. DBA example

```
<?php

$id = dba_open ("/tmp/test.db", "n", "db2");

if (!$id) {
    echo "dba_open failed\n";
    exit;
}

dba_replace ("key", "This is an example!", $id);

if (dba_exists ("key", $id)) {
    echo dba_fetch ("key", $id);
    dba_delete ("key", $id);
}

dba_close ($id);
?>
```

DBA is binary safe and does not have any arbitrary limits. However, it inherits all limits set by the underlying database implementation.

All file-based databases must provide a way of setting the file mode of a new created database, if that is possible at all. The file mode is commonly passed as the fourth argument to **dba_open()** or **dba_popen()**.

You can access all entries of a database in a linear way by using the **dba_firstkey()** and **dba_nextkey()** functions. You may not change the database while traversing it.

Example 2. Traversing a database

```
<?php

// ...open database...

$key = dba_firstkey ($id);

while ($key != false) {
```

```
$key = dba_firstkey ($id);

while ($key != false) {
    if (...) {          // remember the key to perform some action later
        $handle_later[] = $key;
    }
    $key = dba_nextkey ($id);
}

for ($i = 0; $i < count($handle_later); $i++)
    dba_delete ($handle_later[$i], $id);

?>
```

Table of Contents

dba_close

(PHP 3>= 3.0.8, PHP 4)

dba_close -- Close database

Description

void **dba_close** (resource handle)

dba_close() closes the established database and frees all resources specified by handle.

handle is a database handle returned by **dba_open()**.

dba_close() does not return any value.

See also: **dba_open()** and **dba_popen()**

dba_delete

(PHP 3>= 3.0.8, PHP 4)

dba_delete -- Delete entry specified by key

Description

bool **dba_delete** (string key, resource handle)

dba_delete() deletes the entry specified by key from the database specified with handle.

key is the key of the entry which is deleted.

handle is a database handle returned by **dba_open()**.

dba_delete() returns **TRUE** or **FALSE**, if the entry is deleted or not deleted, respectively.

See also: **dba_exists()**, **dba_fetch()**, **dba_insert()**, and **dba_replace()**.

dba_exists

(PHP 3>= 3.0.8, PHP 4)

dba_exists -- Check whether key exists

Description

bool **dba_exists** (string key, resource handle)

dba_exists() checks whether the specified key exists in the database specified by handle.

Key is the key the check is performed for.

Handle is a database handle returned by **dba_open()**.

dba_exists() returns **TRUE** or **FALSE**, if the key is found or not found, respectively.

See also: **dba_fetch()**, **dba_delete()**, **dba_insert()**, and **dba_replace()**.

dba_fetch

(PHP 3>= 3.0.8, PHP 4)

dba_fetch -- Fetch data specified by key

Description

Description

string **dba_fetch** (string key [, int skip, resource handle])

dba_fetch() fetches the data specified by key from the database specified with handle.

Key is the key the data is specified by.

Skip is the number of key-value pairs to ignore when using cdb databases. This value is ignored for all other databases which do not support multiple keys with the same name.

Handle is a database handle returned by **dba_open()**.

dba_fetch() returns the associated string or **FALSE**, if the key/data pair is found or not found, respectively.

See also: **dba_exists()**, **dba_delete()**, **dba_insert()**, and **dba_replace()**.

> **Note:** The skip parameter is available since PHP 4.3 to support cdb's capability of multiple keys having the same name.

dba_firstkey

(PHP 3>= 3.0.8, PHP 4)

dba_firstkey -- Fetch first key

Description

string **dba_firstkey** (resource handle)

dba_firstkey() returns the first key of the database specified by handle and resets the internal key pointer. This permits a linear search through the whole database.

Handle is a database handle returned by **dba_open()**.

dba_firstkey() returns the key or **FALSE** depending on whether it succeeds or fails, respectively.

See also: **dba_nextkey()** and example 2 in the DBA examples

dba_handlers

(PHP 4 >= 4.3.0)

dba_handlers -- List handlers available

Description

array **dba_handlers** (void)

dba_handlers() returns an array with all handlers suppoerted by this extension.

When the internal cdb library is used you will see 'cdb' and 'cdb_make'.

dba_insert

(PHP 3>= 3.0.8, PHP 4)

dba_insert -- Insert entry

Description

bool **dba_insert** (string key, string value, resource handle)

dba_insert() inserts the entry described with key and value into the database specified by handle. It fails, if an entry with the same key already exists.

key is the key of the entry to be inserted.

value is the value to be inserted.

handle is a database handle returned by **dba_open()**.

dba_insert() returns **TRUE** or **FALSE**, depending on whether it succeeds of fails, respectively.

See also: **dba_exists() dba_delete() dba_fetch() dba_replace()**

dba_list

(PHP 4 >= 4.3.0)

dba_list -- List all open database files

Description

array **dba_list** (void)

dba_list() returns an associative array with all open database files. This array is in the form: resourceid=>filename.

dba_nextkey

(PHP 3>= 3.0.8, PHP 4)

(PHP 3>= 3.0.8, PHP 4)

dba_nextkey -- Fetch next key

Description

string **dba_nextkey** (resource handle)

dba_nextkey() returns the next key of the database specified by handle and advances the internal key pointer.

handle is a database handle returned by **dba_open()**.

dba_nextkey() returns the key or **FALSE** depending on whether it succeeds or fails, respectively.

See also: **dba_firstkey()** and example 2 in the DBA examples

dba_open

(PHP 3>= 3.0.8, PHP 4)

dba_open -- Open database

Description

resource **dba_open** (string path, string mode, string handler [, ...])

dba_open() establishes a database instance for path with mode using handler.

path is commonly a regular path in your filesystem.

mode is "r" for read access, "w" for read/write access to an already existing database, "c" for read/write access and database creation if it doesn't currently exist, and "n" for create, truncate and read/write access. Additional you can set the databse lock method with the next char. Use "l" to lock the database with an .lck file or "d" to lock the databasefile itself. It is important that all of your applications do this consistently. If you want to test the access and do not want to wait for the lock you can add "t" as third character. When you are absolutly sure that you do not require database locking you can do so by using "-" instead of "l" or "d". When none of "d", "l" or "-" is used dba will lock on the database file as it would with "d".

handler is the name of the handler which shall be used for accessing path. It is passed all optional parameters given to **dba_open()** and can act on behalf of them.

dba_open() returns a positive handle or **FALSE**, in the case the database was opened successfull or fails, respectively.

> **Note:** There can only be one writer for one database file. When you use dba on a webserver and more than one request requires write operations they can only be done one after another. Also read during write is not allowed. The dba extension uses locks to prevent this. See the follwoing table:

Table 1. DBA locking

already open	mode = "rl"	mode = "rlt"	mode = "wl"	mode = "wlt"	